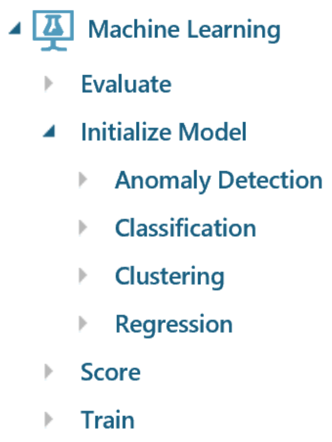
Regression – Under the Hood

- Normal Equations
- Computational Solutions
- Gradient Descent

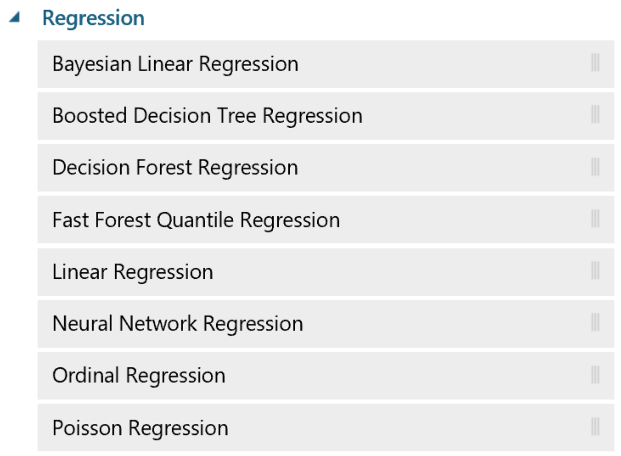
BAUER COLLEGE OF BUSINESS
UNIVERSITY of HOUSTON

Regression

Models *(sample from AML)*



- Machine Learning
  - Evaluate
  - Initialize Model
    - Anomaly Detection
    - Classification
    - Clustering
    - Regression
  - Score
  - Train

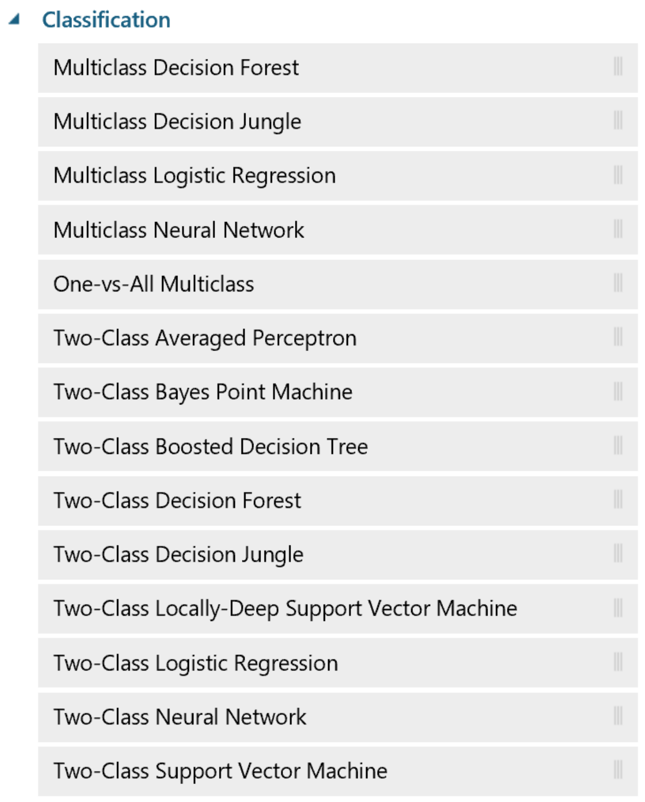


- Regression
  - Bayesian Linear Regression
  - Boosted Decision Tree Regression
  - Decision Forest Regression
  - Fast Forest Quantile Regression
  - Linear Regression
  - Neural Network Regression
  - Ordinal Regression
  - Poisson Regression

CRAN
(too numerous to display)

https://cran.r-project.org/doc/contrib/Ricci-refcard-regression.pdf

see reference card on blackboard



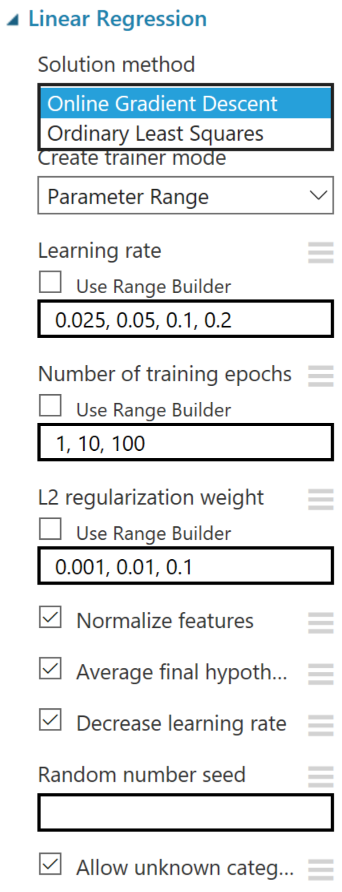
- Classification
  - Multiclass Decision Forest
  - Multiclass Decision Jungle
  - Multiclass Logistic Regression
  - Multiclass Neural Network
  - One-vs-All Multiclass
  - Two-Class Averaged Perceptron
  - Two-Class Bayes Point Machine
  - Two-Class Boosted Decision Tree
  - Two-Class Decision Forest
  - Two-Class Decision Jungle
  - Two-Class Locally-Deep Support Vector Machine
  - Two-Class Logistic Regression
  - Two-Class Neural Network
  - Two-Class Support Vector Machine

CRAN
(too numerous to display)

BAUER COLLEGE OF BUSINESS
UNIVERSITY of HOUSTON

Regression

Parameters *(sample from AML)*



Linear Regression

Solution method

- [✓ Online Gradient Descent]
- Ordinary Least Squares

Create trainer mode: Parameter Range

Learning rate
☐ Use Range Builder
0.025, 0.05, 0.1, 0.2

Number of training epochs
☐ Use Range Builder
1, 10, 100

L2 regularization weight
☐ Use Range Builder
0.001, 0.01, 0.1

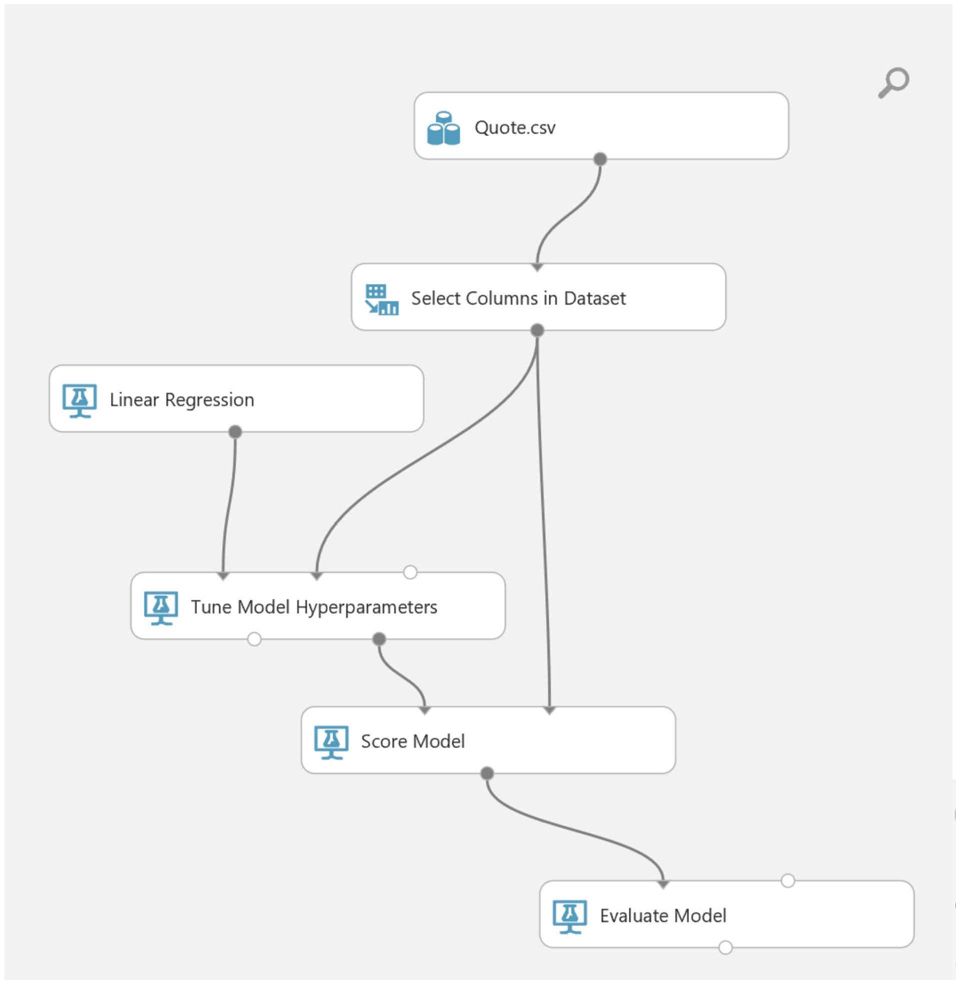
☑ Normalize features

☑ Average final hypoth...

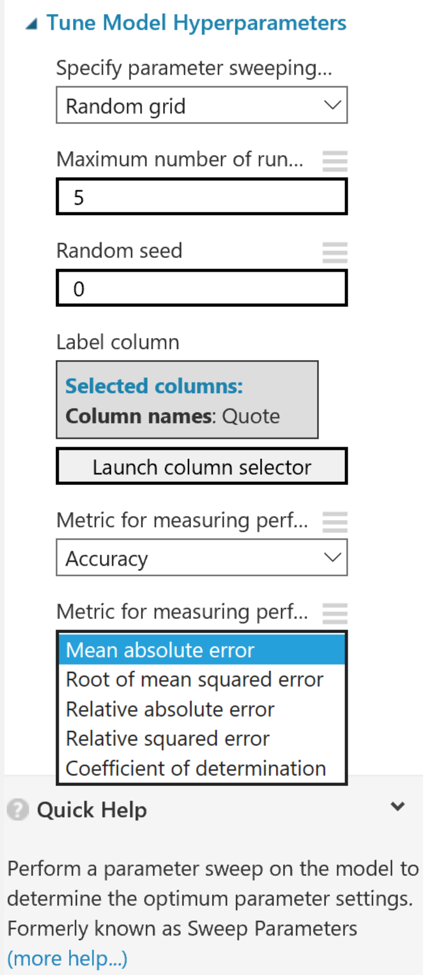
☑ Decrease learning rate

Random number seed

☑ Allow unknown categ...



Evaluation



Tune Model Hyperparameters

Specify parameter sweeping...: Random grid

Maximum number of run...: 5

Random seed: 0

Label column

Selected columns:
Column names: Quote

Launch column selector

Metric for measuring perf...: Accuracy

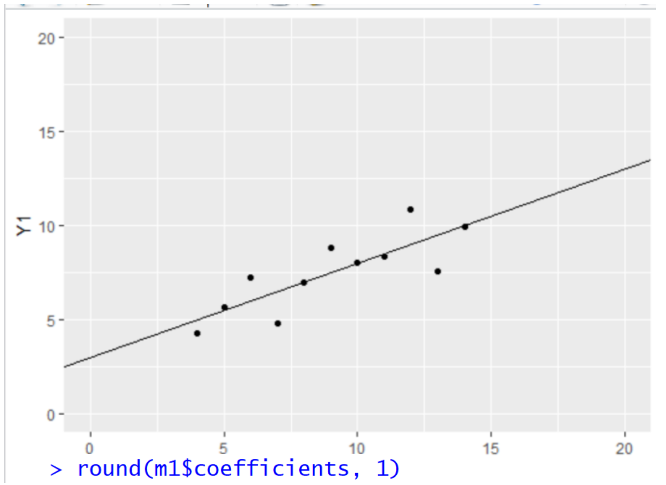
Metric for measuring perf...

- [✓ Mean absolute error]
- Root of mean squared error
- Relative absolute error
- Relative squared error
- Coefficient of determination

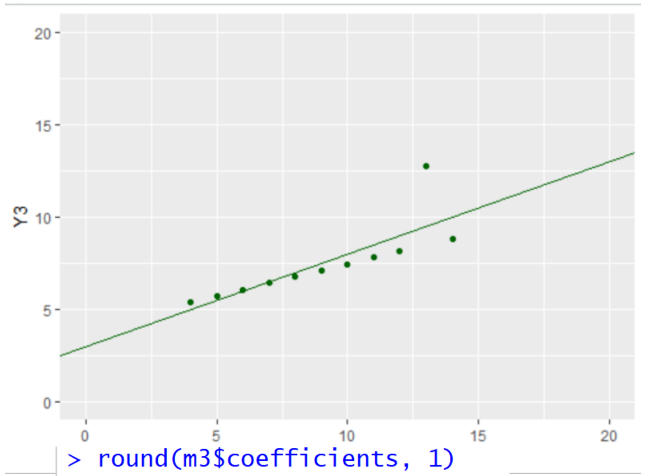
Quick Help

Perform a parameter sweep on the model to determine the optimum parameter settings. Formerly known as Sweep Parameters
(more help...)

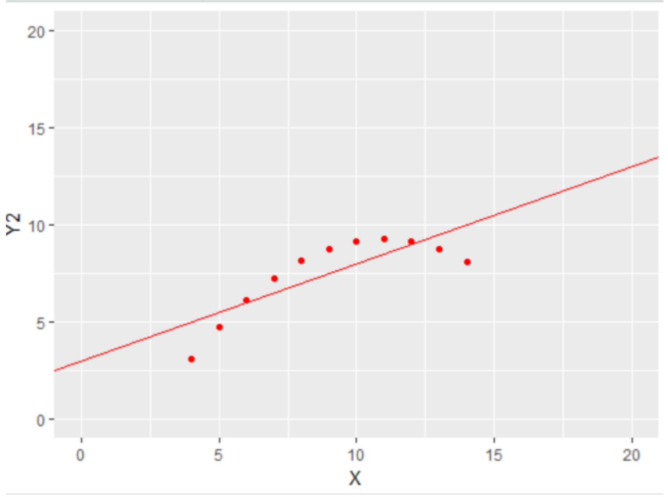
Anscombe's Quartet



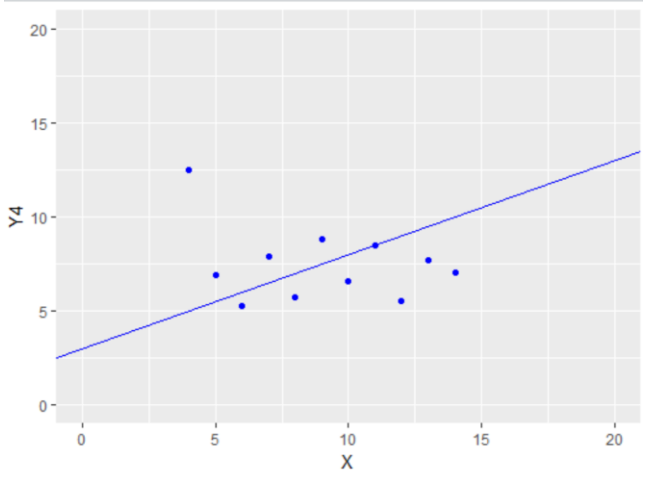
```
> round(m1$coefficients, 1)
(Intercept)           X
        3.0         0.5
```



```
> round(m3$coefficients, 1)
(Intercept)           X
        3.0         0.5
```



```
> round(m2$coefficients, 1)
(Intercept)           X
        3.0         0.5
>
```



```
> round(m4$coefficients, 1)
(Intercept)          X4
        3.0         0.5
```

How does this speak to the bias variance tradeoff?

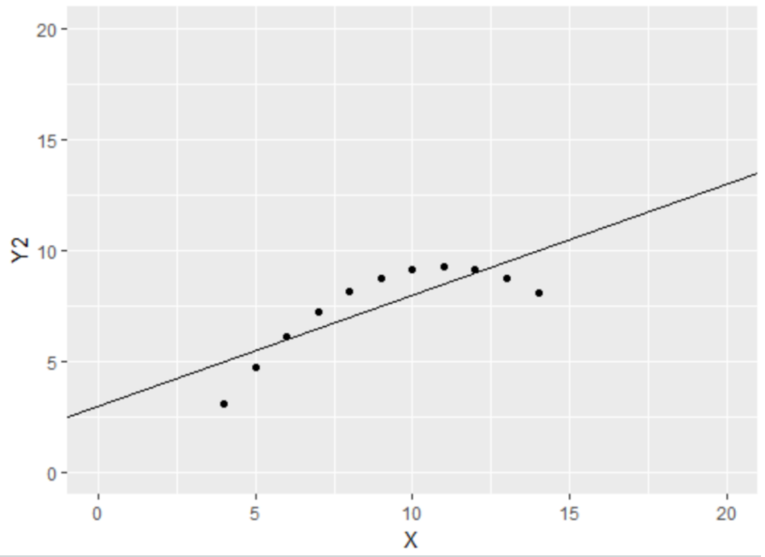
How do outliers affect the linear model?

How well does a linear model reflect data?

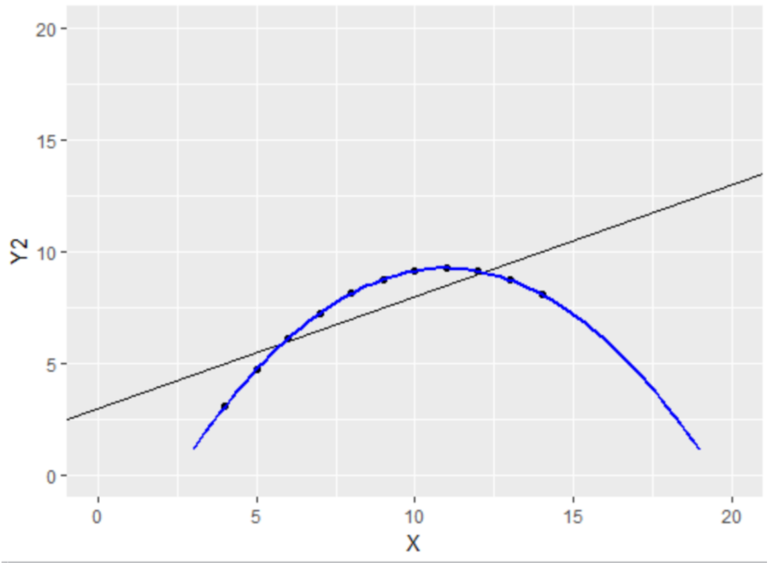
Recall the assumptions of the linear model from the Airlift

Linearity
Homoscedasticity
N

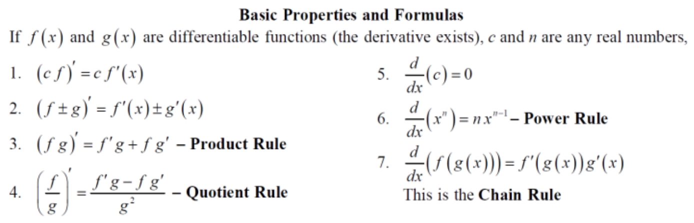
Using the linear model:
m2 <- lm(Y2 ~ X, data = tst)



Using a polynomial model:
lfit <- lm(Y2 ~ X + I(X^2), tst2)



Calculus cheat sheet (blackboard)



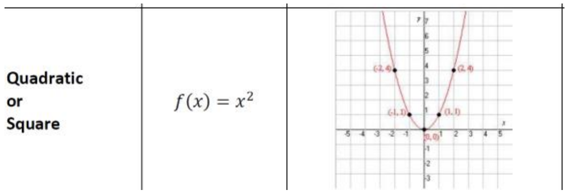
Basic Properties and Formulas

If $f(x)$ and $g(x)$ are differentiable functions (the derivative exists), c and n are any real numbers,

1. $(cf)' = cf'(x)$
2. $(f \pm g)' = f'(x) \pm g'(x)$
3. $(fg)' = f'g + fg'$ – **Product Rule**
4. $\left(\frac{f}{g}\right)' = \frac{f'g - fg'}{g^2}$ – **Quotient Rule**
5. $\frac{d}{dx}(c) = 0$
6. $\frac{d}{dx}(x^n) = nx^{n-1}$ – **Power Rule**
7. $\frac{d}{dx}\left(f(g(x))\right) = f'(g(x))g'(x)$

This is the **Chain Rule**

Harold's cheat sheet (blackboard)



Quadratic or Square	$f(x) = x^2$	

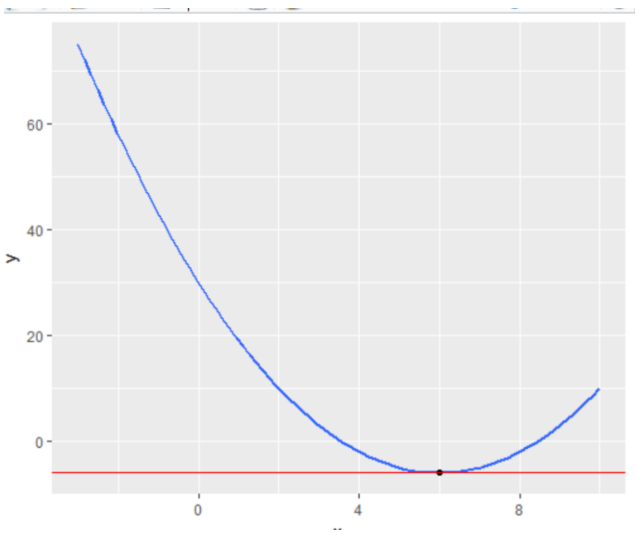
(-2,4)
(2,4)
(-1,1)
(1,1)
(0,0)

Recall: the first derivative gets us the tangent, setting slope of the tangent to 0 gets us to the max or min of a convex function.

$x^2 - 12x - 10$

$0 = 2x - 12$

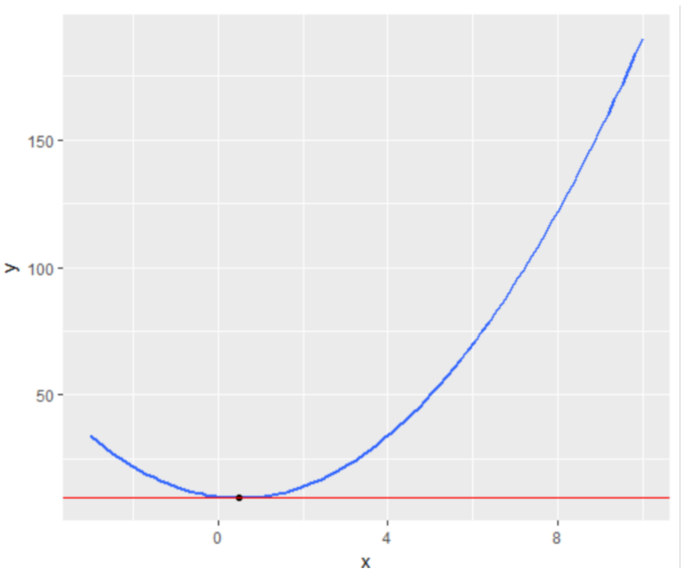
$x = -6$



$2x^2 - 2x + 10$

$0 = 4x - 2$

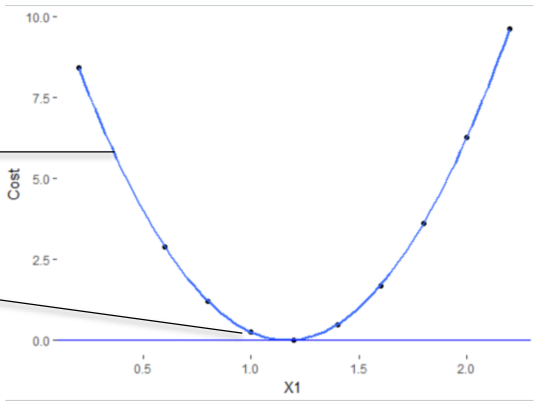
$x = 0.5$

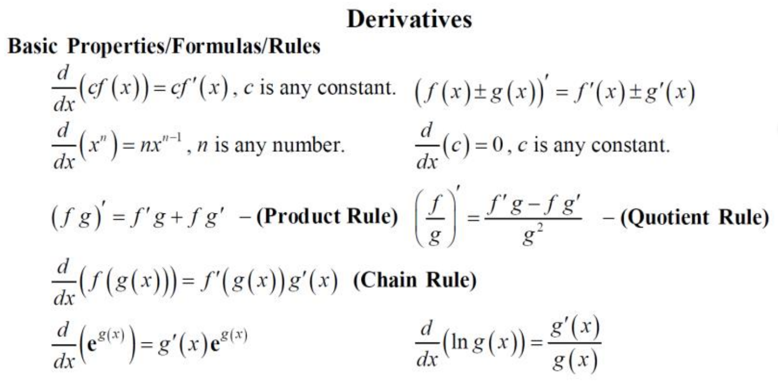


Analytic Solutions:

Cost Function (assumed to be convex)

Minimum = first derivative set to 0





Derivatives

Basic Properties/Formulas/Rules

$\frac{d}{dx}(cf(x)) = cf'(x)$, c is any constant. $(f(x) \pm g(x))' = f'(x) \pm g'(x)$

$\frac{d}{dx}(x^n) = nx^{n-1}$, n is any number. $\frac{d}{dx}(c) = 0$, c is any constant.

$(fg)' = f'g + fg'$ – **(Product Rule)** $\left(\frac{f}{g}\right)' = \frac{f'g - fg'}{g^2}$ – **(Quotient Rule)**

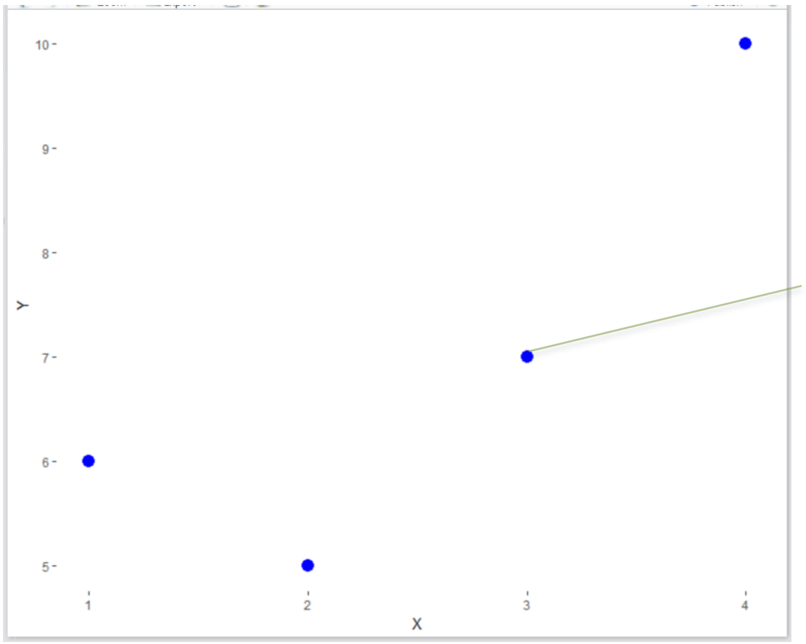
$\frac{d}{dx}(f(g(x))) = f'(g(x))g'(x)$ **(Chain Rule)**

$\frac{d}{dx}\left(\mathbf{e}^{g(x)}\right) = g'(x)\mathbf{e}^{g(x)}$ $\frac{d}{dx}(\ln g(x)) = \frac{g'(x)}{g(x)}$

See:
common_derivatives_integrals_cheatsheet

In Blackboard

free book on convex optimization, if interested: https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf

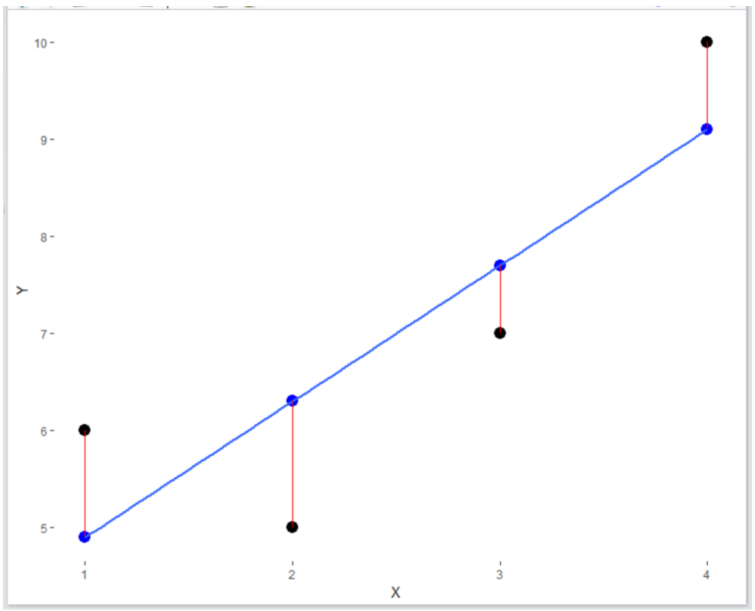


Taken from Wikipedia article (good!)
Also see common derivatives_integrals cheat sheet

$\beta_0 + 1\beta_1$ = 6
$\beta_0 + 2\beta_1$ = 5
$\beta_0 + 3\beta_1$ = 7
$\beta_0 + 4\beta_1$ = 10

$S(\beta_1, \beta_2\ ) =$ $[6- (\beta_0 +\ 1\beta_1\ )]^2$
$[5- (\beta_0 +\ 2\beta_1\ )]^2$
$[7- (\beta_0 +\ 3\beta_1\ )]^2$
$[10- (\beta_0 +\ 4\beta_1\ )]^2$

$$= 4\beta_0^2 + 20\beta_0\beta_1 \ - 56\beta_0 + 30\beta_1^2 - 154\beta_1 \ + 210$$



$$\frac{\partial s}{\partial \beta_0} = \ 8\beta_0 + 20\beta_1 \qquad = 56$$

$$\frac{\partial s}{\partial \beta_1} = \ 20\beta_0 + 60\beta_1 \quad = 154$$

analytic derivative based solutions become intractable in the real world

BAUER COLLEGE OF BUSINESS UNIVERSITY of HOUSTON

Solving with normal equations *(from LA Review)*

Solving the derivative equations

$$8\beta_1 + 20\beta_2 = 56$$
$$20\beta_1 + 60\beta_2 = 154$$

```
solve(X, B)
[1,] 3.5
[2,] 1.4
```

```
X <- matrix( c(8, 20, 20, 60), nrow=2, ncol = 2)
B <- matrix( c(56, 154), nrow=2, ncol = 1)
solve(X, B)
```

Remember, solve will give an inverse matrix if you give it one matrix as an input, and will solve a system of equations if you give it 2 matrices

Solvig the normal equations

$$\beta = (X^T X)^{-1} (X^T Y)$$

```
X <- cbind(1, mydata$X)
y <- mydata$Y
# we can solve this from the raw data by using a transpose
betaHat <- solve(t(X)%*%X) %*% t(X) %*%y
print(betaHat)
```

```
print(betaHat)
[1,] 3.5
[2,] 1.4
```

[recall from LA review – solve gives you the inverse matrix]

Solving using single value decomposition

```
# now solving using SVD

x <- t(X) %*% X
duv <- svd(x)
x.inv <- duv$v %*% diag(1 / duv$d) %*% t(duv$u)
x.pseudo.inv <- x.inv %*% t(X)
w <- x.pseudo.inv %*% y
w
```

```
w
[1,] 3.5
[2,] 1.4
```

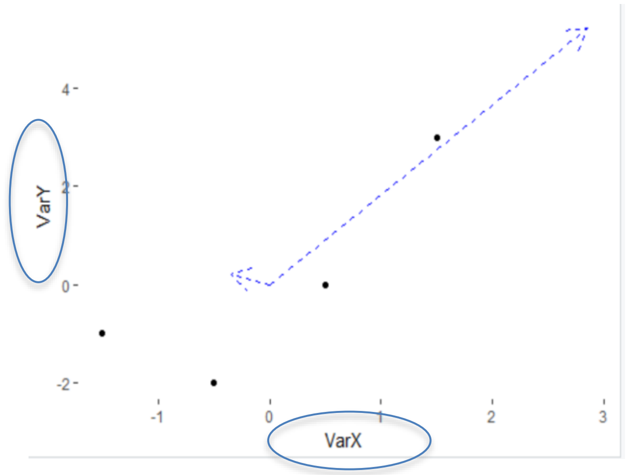
```
# note we can also use SVD for dimension reduction (like PCA)
# it's also used in advanced numerical solutions (won't be doing that here)
```

We will look at another matrix decomposition method (QR) soon.

Solving from variance – covariance matrix

Recall from PCA analysis in DA1, the variance and covariance matrix:

X	Y	Var X	Var Y	covXY
1	6	-1.50	-1.00	1.50
2	5	-0.50	-2.00	1.00
3	7	0.50	0.00	0.00
4	10	1.50	3.00	4.50
Total		**0.00**	**0.00**	**0.00**



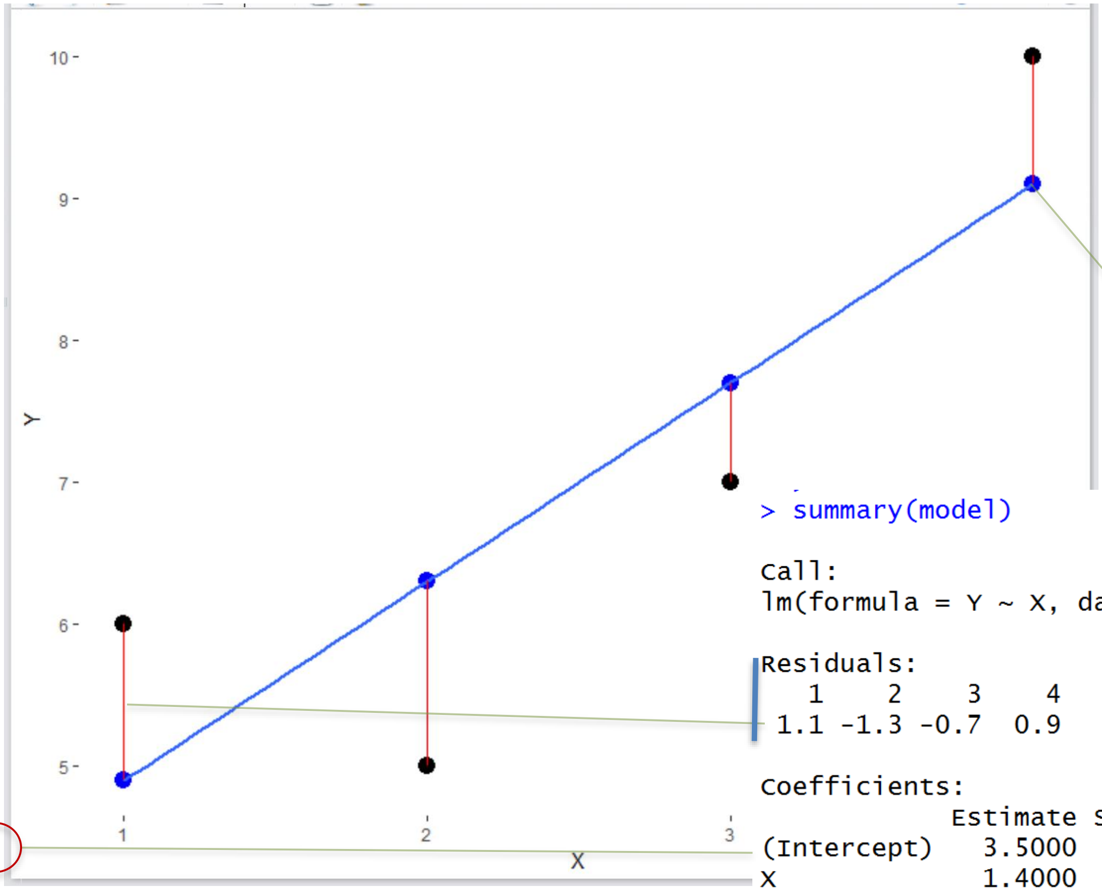
Using covariance to get eigenvectors
(see code file)

```
# and we can use the covaraince matrices to get the estimated coefficients

b1 <- solve(cov(pcadata$X, pcadata$X), cov(pcadata$X, pcadata$Y))
mean(pcadata$Y - (pcadata$X * b1))
```

So to review, we can use the variance/covariance matrices to estimate coefficients and we can apply eigenvectors to describe the variance

Or… we can use lm *(which uses QR decomposition* to solve.

lm is a very comprehensive function and supplies a list of metrics and diagnostics.



```
> fitted(model)
  1   2   3   4 
4.9 6.3 7.7 9.1 
```

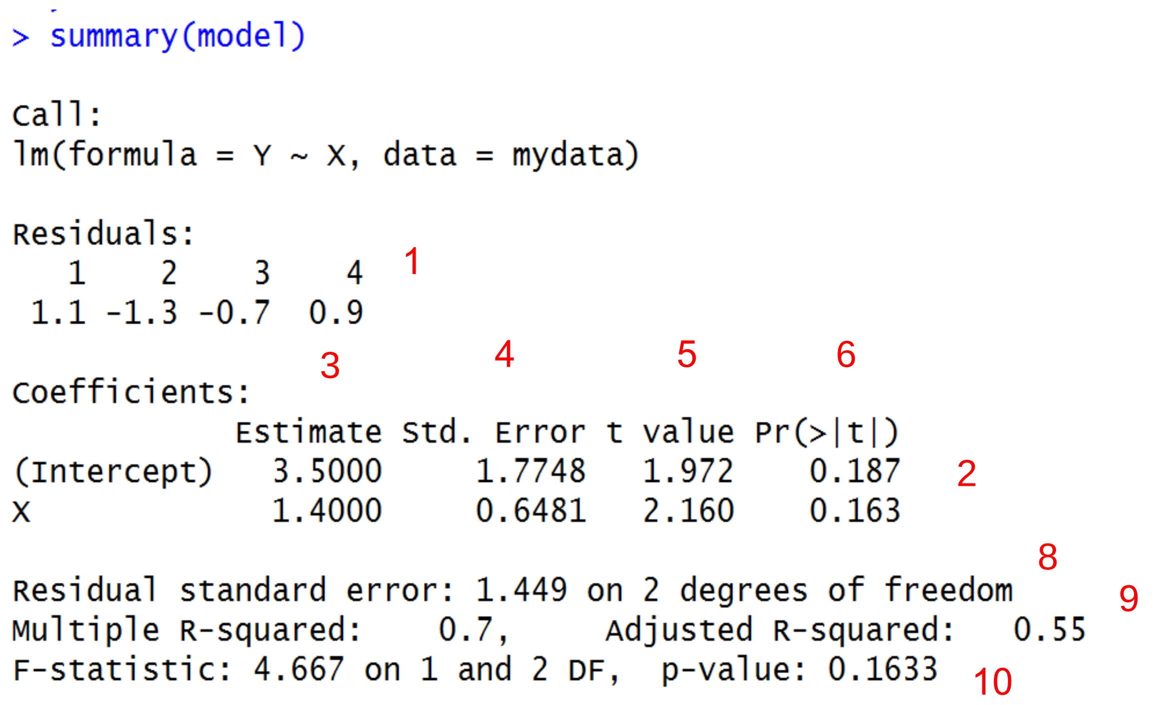
```
> summary(model)

Call:
lm(formula = Y ~ X, data = mydata)

Residuals:
   1    2    3    4 
 1.1 -1.3 -0.7  0.9 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   3.5000     1.7748   1.972    0.187
X             1.4000     0.6481   2.160    0.163

Residual standard error: 1.449 on 2 degrees of freedom
Multiple R-squared:    0.7,	Adjusted R-squared:   0.55 
F-statistic: 4.667 on 1 and 2 DF,  p-value: 0.1633
```

The residual is the difference between the value of the dependent variable predicted by the model, and the true value of the dependent variable. One method of estimation is ordinary least squares.

R lm summary

Once the coefficients are estimated, we have a whole new set of covariances to analyze.

…legend on next slide



```
> summary(model)

Call:
lm(formula = Y ~ X, data = mydata)

Residuals:
   1    2    3    4    1
 1.1 -1.3 -0.7  0.9

                    3         4          5        6
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   3.5000     1.7748   1.972    0.187    2
X             1.4000     0.6481   2.160    0.163
                                                       8
Residual standard error: 1.449 on 2 degrees of freedom     9
Multiple R-squared:    0.7,     Adjusted R-squared:   0.55
F-statistic: 4.667 on 1 and 2 DF,  p-value: 0.1633   10
```

#	Name	Description
1	Residuals	y – ŷ. If our residuals are normally distributed, this indicates the mean of the difference between our predictions and the actual values is close to 0 (good).
2	Significance Stars	The stars are shorthand for significance levels (p-value) computed. *** for high significance and * for low significance.
3	Estimated Coefficient	The estimated coefficient is the value of slope calculated by the regression (think of the Intercept as a slope that is always multiplied by 1). Always good to spot check this number to make sure it seems reasonable.
4	Standard Error of the Coefficient	Measure of the variability in the estimate for the coefficient. Lower means better but this number is relative to the value of the coefficient. As a rule of thumb, you'd like this value to be at least an order of magnitude less than the coefficient estimate.
5	t-value of the Coefficient Estimate	Score that measures whether or not the coefficient for this variable is meaningful for the model. Used to calculate the p-value and the significance levels.
6	Variable p-value	Probability the variable is NOT relevant. You want this number to be as small as possible. If the number is really small, R will display it in scientific notation. In or example 2e-16 means that the odds that parent is meaningless is about $^1/_{5000000000000000}$
7	Significance Legend	The more punctuation there is next to your variables, the better. Blank=bad, Dots=pretty good, Stars=good, More Stars=very good
8	Residual Std Error / Degrees of Freedom	Standard deviation of your residuals. You'd like this number to be proportional to the quantiles of the residuals. For a normal distribution, the 1st and 3rd quantiles should be 1.5 +/- the std error. The *Degrees of Freedom* is the difference between the number of observations included in your training sample and the number of variables used in your model *(intercept counts as a variable)*. Degree of freedom is the number of non-zero coefficient estimates
9	R-squared	Metric for evaluating the goodness of fit of your model. Higher is better with 1 being the best. Corresponds with the amount of variability in what you're predicting that is explained by the model. In this instance.

```
> summary(model)

Call:
lm(formula = Y ~ X, data = mydata)

Residuals:
   1    2    3    4 
 1.1 -1.3 -0.7  0.9 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   3.5000     1.7748   1.972    0.187
X             1.4000     0.6481   2.160    0.163

Residual standard error: 1.449 on 2 degrees of freedom
Multiple R-squared:    0.7,	Adjusted R-squared:   0.55 
F-statistic: 4.667 on 1 and 2 DF,  p-value: 0.1633
```

Dimension Level

SSR			
Y Hat	Mean Y	Var	Sum Sq
4.9	7	-2.10	4.41
6.3	7	-0.70	0.49
7.7	7	0.70	0.49
9.1	7	2.10	4.41
Total		**0.00**	**9.80**

SSE			
Y	Y Hat	Var	Sum Sq
6	4.9	1.10	1.21
5	6.3	-1.30	1.69
7	7.7	-0.70	0.49
10	9.1	0.90	0.81
Total		**0.00**	**4.20**

Degrees of freedom: df = n-k-1

n = number of observations

k = number of independent (explanatory) variables

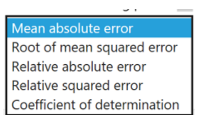
Model Level

SST (SSR + SSE)	14.00
R^2 (SSR/SST)	0.70
MST (SST / SST df))	4.67
MSE (SSE / (SSE df))	2.10
Adj R^2 (1-MSE/MST)	0.55
RSE (SQRT(SSE/df))	1.45

$SSR = \sum (\hat{y}_i - \bar{y})^2$ = sum of squares due to the regression

$SSE = \sum (y_i - \hat{y}_i)^2$ = sum of squares due to error – also called the residual sum of squares

$SST = SSR + SSE$



Azure regression diagnostic metrics

$$\frac{|\hat{y} - y|}{n}$$

Mean Absolute Error *(MAE)* Same unit as the original data, so it can *only be compared* ***between models whose errors are measured in the same units***. Easy to understand. Each error influences MAE in direct proportion to the absolute value of the error, which is not the case for RMSE. **MAE**

$$\sqrt{MSE}$$

Root of mean squared error (RMSE) *(same thing as residual standard error RSE – yes, I know)* . Again, for comparing models measure in the **same units**. ***RMSE amplifies large errors.***

$$\frac{|\hat{y} - y|}{|\bar{y} - y|}$$

RAE can be compared between models whose errors are measured in the ***different units*** *(why would we use RAE?). We divide the absolute those differences by the error so they have a scale from 0 to 1 (and if you multiply this value by 100 you get a percentage)*

$$\frac{(\hat{y} - y)^2}{(\bar{y} - y)^2}$$

RSE Relative Squared Error **can be compared between models** whose errors are measured in the different units

R^2 the coefficient of determination (R^2) summarizes the explanatory power of the regression model. If the regression model is perfect R^2 is 1. If the regression model is a total failure, R2 is zero.

So, now that we have 20+ freakin' diagnostic metrics just for regression alone (and we're not nearly done), what do we use when and where?

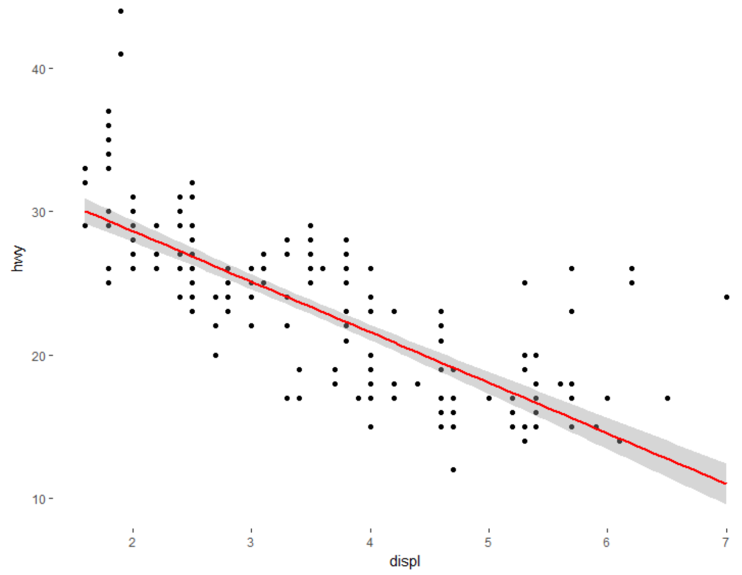
And the answer is… it depends

Confidence Intervals

Assumptions of Linear Regression

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i, \quad i = 1, \ldots, n.$$

- Linearity *(explanatory and response variables are linearly related)*
- Independence (*little or no correlation in the explanatory variables)*
- Constant Variance *(homoscedasticity – variance is evenly distributed)*
- Normality *(residuals are normally distributed)*



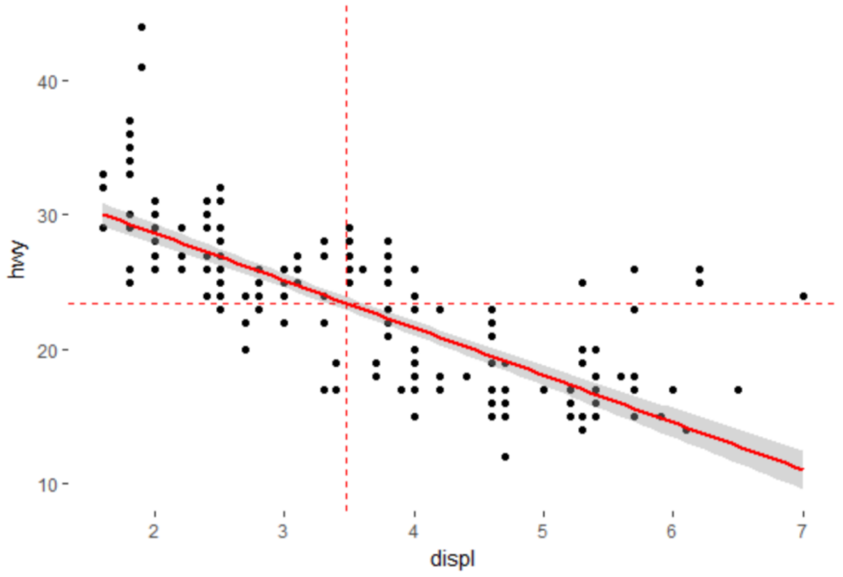
```
library(tidyverse)
dfMPG <- mpg

# chart the mpg data

p <- ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  geom_smooth(method = lm, mapping = aes(x = displ, y = hwy), 
col = "red")+
  theme(
    panel.background = element_rect(fill = "white") # I do this so 
it's easier to see in  class
  )
p
```

For normal distributions, 95% means (μ) will lie between $\pm$ 2SE

Confidence Interval : prediction $\pm$ 2SE

```
# add the means to the chart

ym <- mean(mpg$hwy)
xm <- mean(mpg$displ)
p <- p+ geom_hline(yintercept=ym, linetype="dashed", color =
"red")
p <- p+ geom_vline(xintercept=xm, linetype="dashed", color =
"red")
p
```

(note calculating the SE is more complex with multiple coefficients – see that section for calculation)

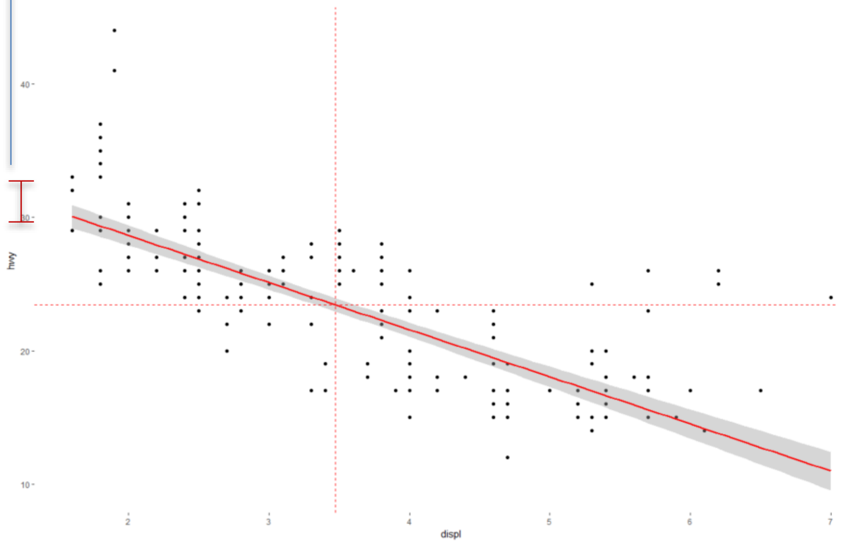


```
# this illustrates how sample size and CI relate
# adjust sample size and resampling to show effect on CI on
slope

masterMod <- lm(data = mpg, hwy~displ) # get the full population
summary(masterMod) # display std errors
SEI <- summary(masterMod)$coefficients[1,2] # get the SE for
the Intercept
MLI <- masterMod$coefficients[1] - (2*SEI) # set a point for lower
CI
MUI <- masterMod$coefficients[1] + (2*SEI) # set a point for
upper CI
MLI
MUI
n <- 20
# number of resampling iterations
resample <- 100

library(tidyverse)
indexes = sample(1:nrow(mpg), n)
mpg2 <- mpg[indexes,]

p <- ggplot(data=mpg2, aes(x=displ, y=hwy)) +
  geom_smooth(method=lm, formula = y ~ x, color = "gray",
se=FALSE) +
  theme(
    panel.background = element_rect(fill = "white")
  )+
  scale_y_continuous(limits = c(0, 50))
p
```

We'll gradually increase the sample size

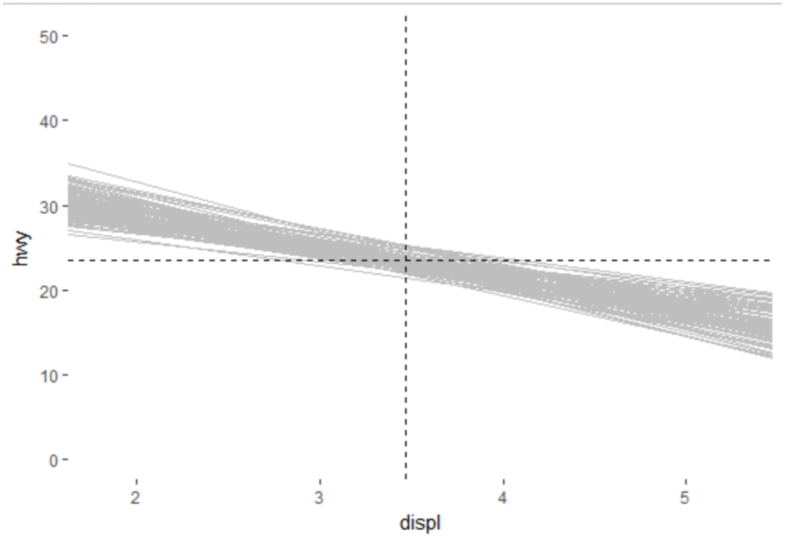
```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  35.6977     0.7204   49.55   <2e-16 ***
displ        -3.5306     0.1945  -18.15   <2e-16 ***
```

```
> MLI
(Intercept)
   34.25692
> MUI
(Intercept)
   37.13839
```



```
Modcoef <- matrix(NA, nrow = resample, ncol = 3)
icnt <- 1
while (icnt <= resample)
{
  indexes = sample(1:nrow(mpg), n)
  mpg2 <- mpg[indexes,]
  mod <- lm(data = mpg2, hwy~displ)
  Modcoef[icnt, 1] <- mod$coefficients[1]
  Modcoef[icnt, 2] <- mod$coefficients[2]
  Modcoef[icnt, 3] <- summary(mod)$coefficients[1,2]
  p <- p +
        geom_abline(intercept = Modcoef[icnt, 1], slope =
Modcoef[icnt, 2], color = "gray")
      icnt <- icnt+1;
}
p

ym <- mean(mpg$hwy)
xm <- mean(mpg$displ)
p <- p+ geom_hline(yintercept=ym, linetype="dashed")
p <- p+ geom_vline(xintercept=xm, linetype="dashed")
p

dfModcoef <- data.frame (Modcoef)

CITest <- nrow(dfModcoef %>% filter(X1 < MUI & X1 > MLI))
```
CITest/resample

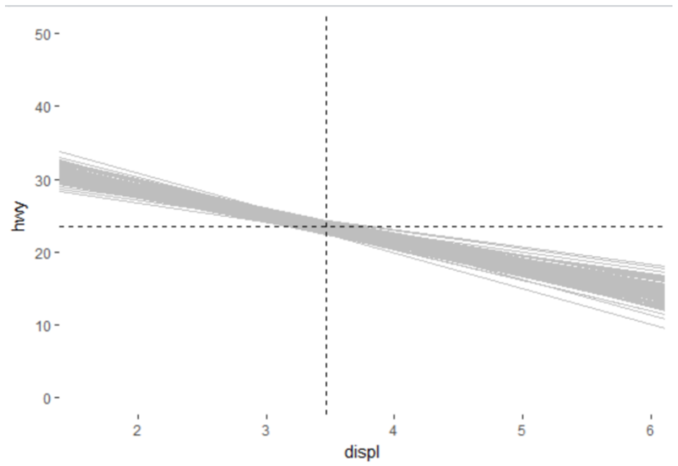
n <- 20



```
> CITest/resample
[1] 0.43
```

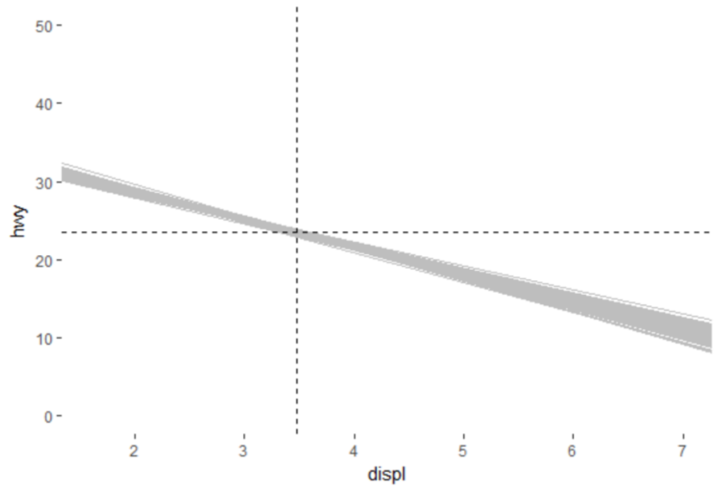
n <- 50

```
> CITest/resample
[1] 0.62
```

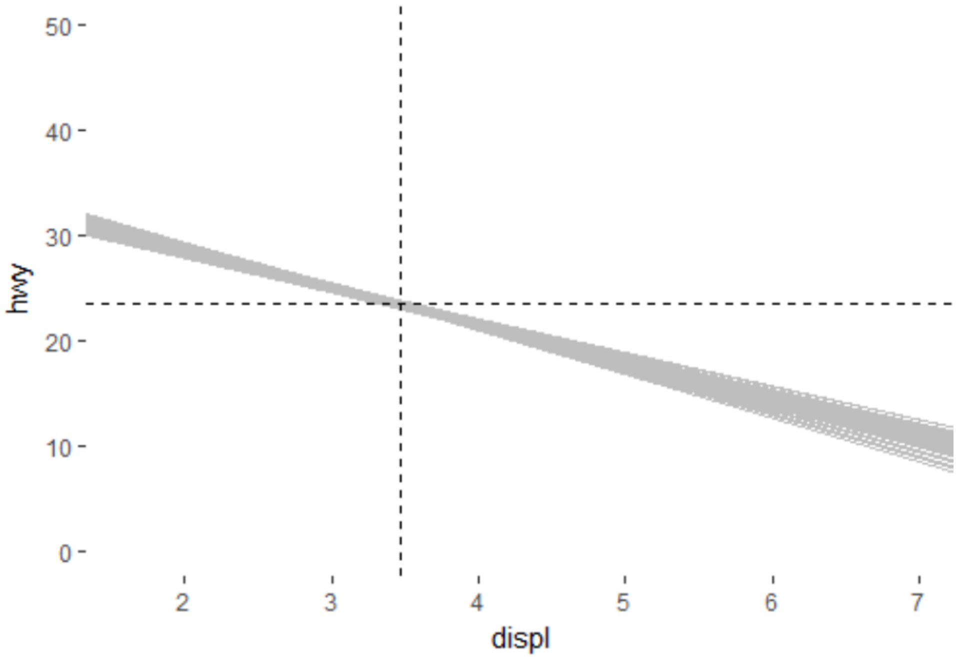


n <- 150

```
> CITest/resample
[1] 0.99
```



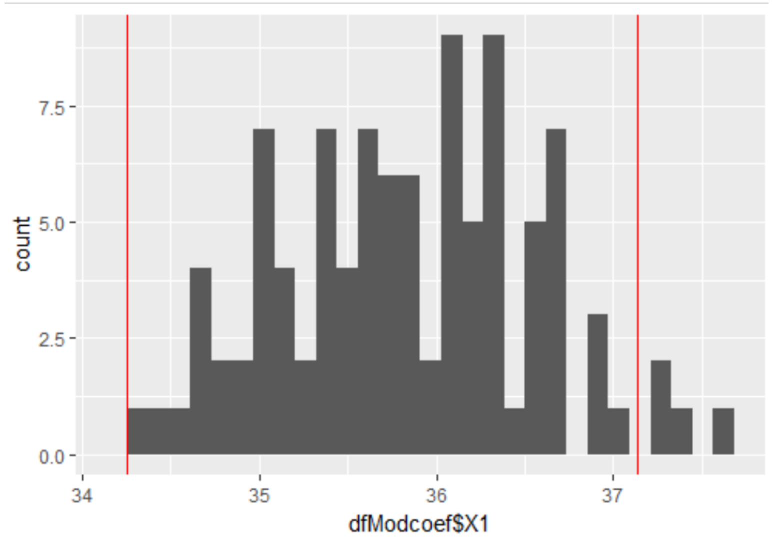
```
# sample size
n <- 150
# number of resampling iterations
resample <- 100

library(tidyverse)
indexes = sample(1:nrow(mpg), n)
mpg2 <- mpg[indexes,]

p <- ggplot(data=mpg2, aes(x=displ, y=hwy)) +
  geom_smooth(method=lm, formula = y ~ x, color = "gray", se=FALSE) +
  theme(
    panel.background = element_rect(fill = "white")
  )+
  scale_y_continuous(limits = c(0, 50))
p

Modcoef <- matrix(NA, nrow = resample, ncol = 3)
icnt <- 1
while (icnt <= resample)
{
  indexes = sample(1:nrow(mpg), n)
  mpg2 <- mpg[indexes,]
  mod <- lm(data = mpg2, hwy~displ)
  Modcoef[icnt, 1] <- mod$coefficients[1]
  Modcoef[icnt, 2] <- mod$coefficients[2]
  Modcoef[icnt, 3] <- summary(mod)$coefficients[1,2]
  p <- p +
       geom_abline(intercept = Modcoef[icnt, 1], slope = Modcoef[icnt, 2], color
= "gray")
     icnt <- icnt+1;
}
p

ym <- mean(mpg$hwy)
xm <- mean(mpg$displ)
```



```
p <- p+ geom_hline(yintercept=ym, linetype="dashed")
p <- p+ geom_vline(xintercept=xm, linetype="dashed")
p
```

```
dfModcoef <- data.frame (Modcoef)

CITest <- nrow(dfModcoef %>% filter(X1 < MUI & X1 > MLI))
CITest/resample

ggplot(data = dfModcoef, aes(dfModcoef$X1)) + geom_histogram() +
  geom_vline(xintercept = MUI, color = 'red') +
  geom_vline(xintercept = MLI, color = 'red')
```

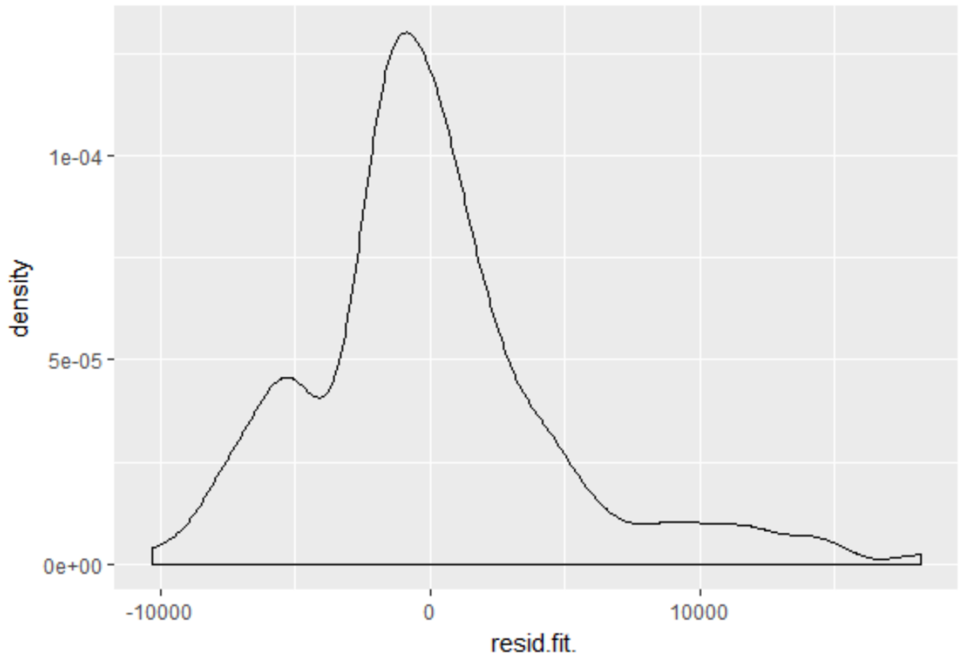
```
> CITest/resample
[1] 0.96
```



Testing Linear Model Assumptions – Normal Distribution of Residuals

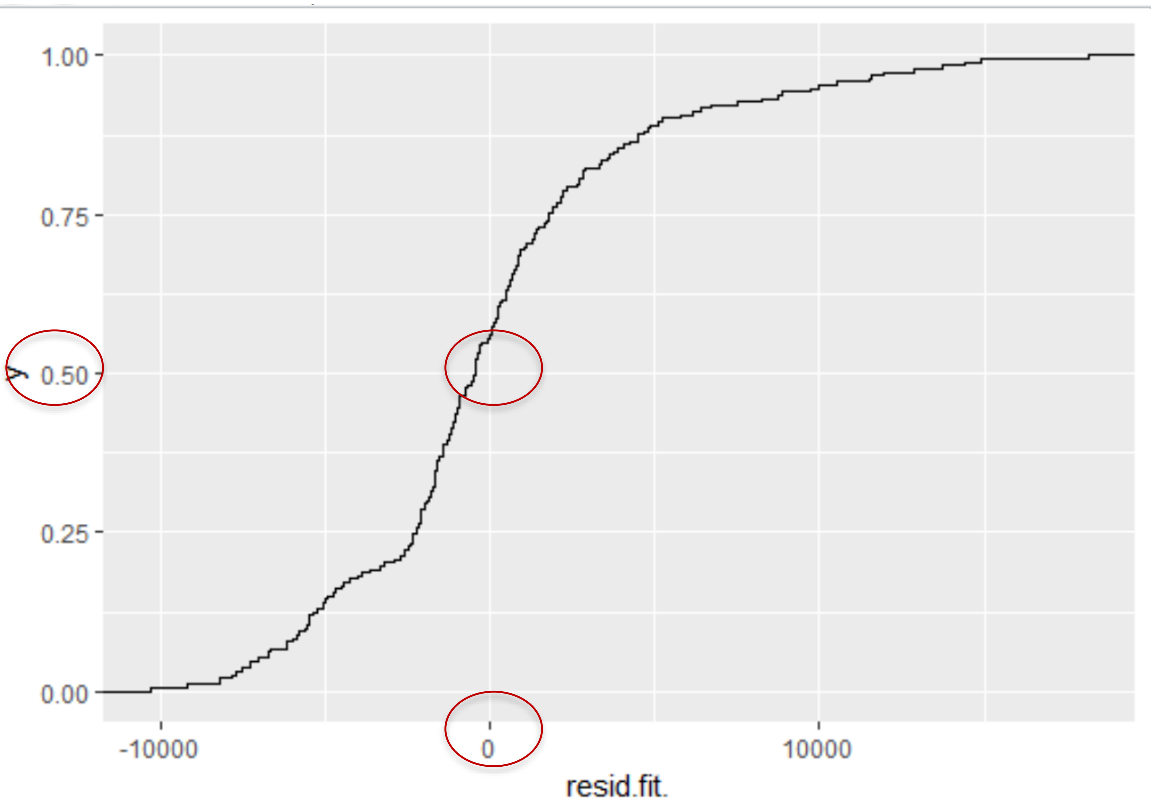
```
p <- ggplot(Auto, aes(x=horsepower, y=price))+geom_point()
p

fit <- lm(price ~ horsepower, data = Auto)

fit.res <- data.frame(resid(fit))
ggplot(data = fit.res, aes(resid.fit.)) +
  geom_density()
result <- shapiro.test(fit.res$resid.fit.)
result$p.value
```

```
> result$p.value
[1] 1.630661e-07
```



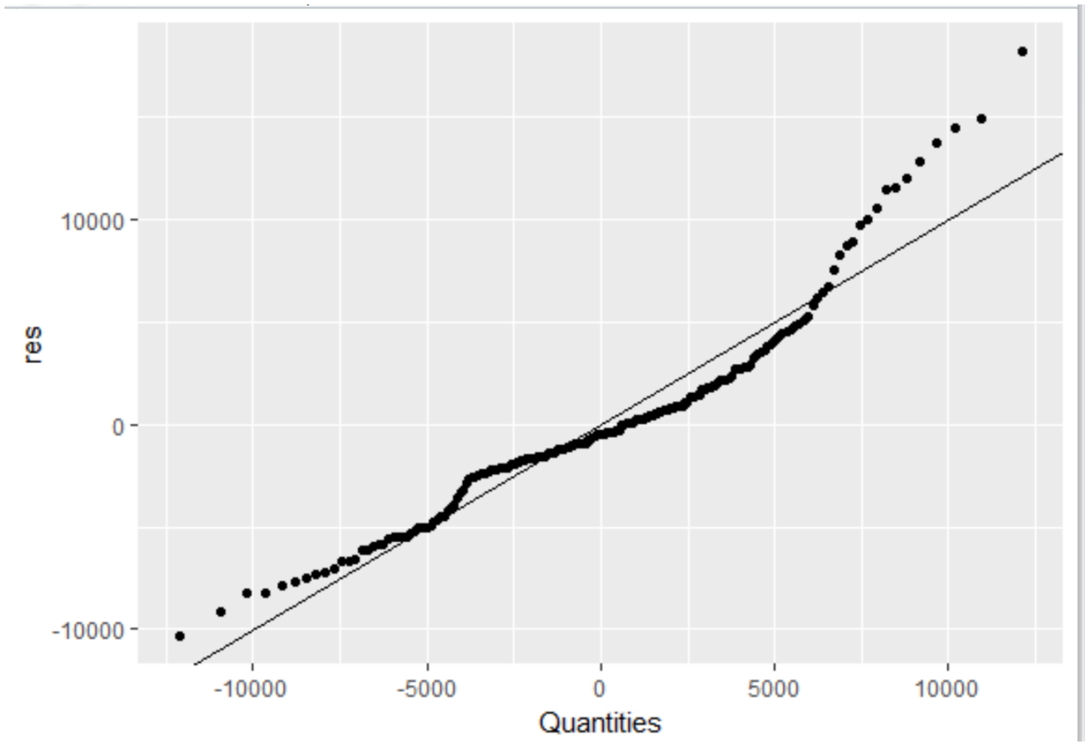
Just says you cannon reject the null hypothesis that the residuals are normally distributed.

```
ggplot(fit.res, aes(resid.fit., ..density..)) +
geom_freqpoly(binwidth = 5000)
ggplot(fit.res, aes(resid.fit., ..density..)) +
geom_histogram(binwidth = 500)
ggplot(fit.res, aes(sample = resid.fit.)) +
stat_qq()
ggplot(fit.res, aes(resid.fit.)) + stat_ecdf()

res <- sort(fit.res$resid.fit.)
res <- res[!is.na(res)]
n = length(res)
```



Just like we've done before, check to see where 50% mark. In the case of residuals, you want the mean to be 0

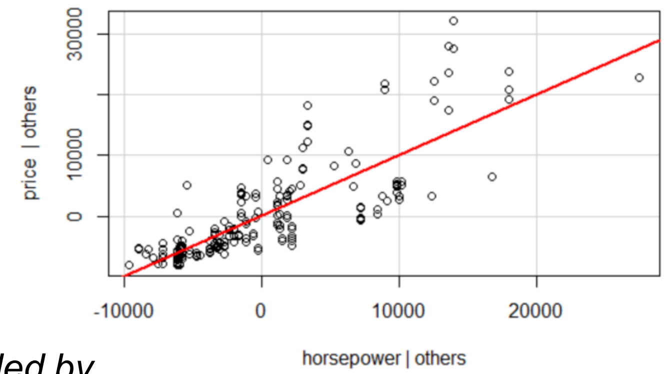
```
probabilities <- (1:n)/(n+1)

# the qnorm fuction creates the theoretical quantities

Quantities <- qnorm(probabilities, mean(res, na.rm = T), sd(res, na.rm = T))

dfProb <- data.frame(cbind(Quantities, res))

p <- ggplot(dfProb, aes(x=Quantities, y = res)) + geom_point()
p
p <- p + geom_abline(intercept = 0, slope = 1)
p
```



Outliers

outlierTest(fit)

```
> outlierTest(fit)
   rstudent unadjusted p-value Bonferonni p
66 4.054475          7.3249e-05     0.014137
```



the p-value reported is for the ***largest*** absolute ***studentized*** *(divided by standard deviation)* residual, using the t distribution with degrees of freedom one less than the residual df for the model

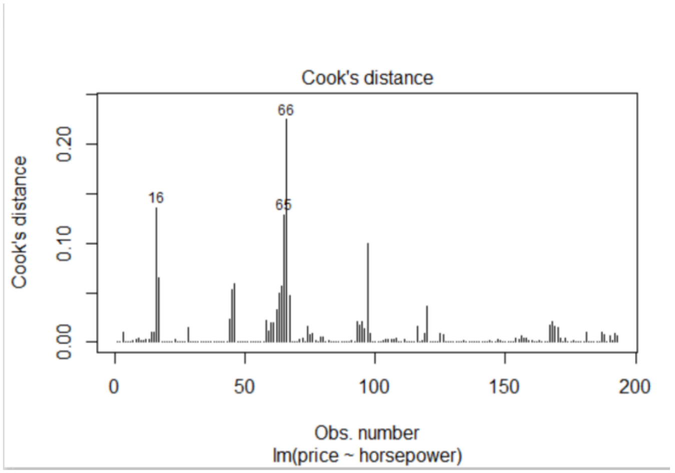
Basically tells you if the outliers are affecting the significance of your model *(still less than the magical .05 here, so 'no' does not indicate that you have to stay up all night screwing with outliers)*

William Sealy Gosset (early 1900s) worte under the pseudonym "Student". Gosset worked at the Guinness Brewery in Dublin, Ireland

Influential Observations

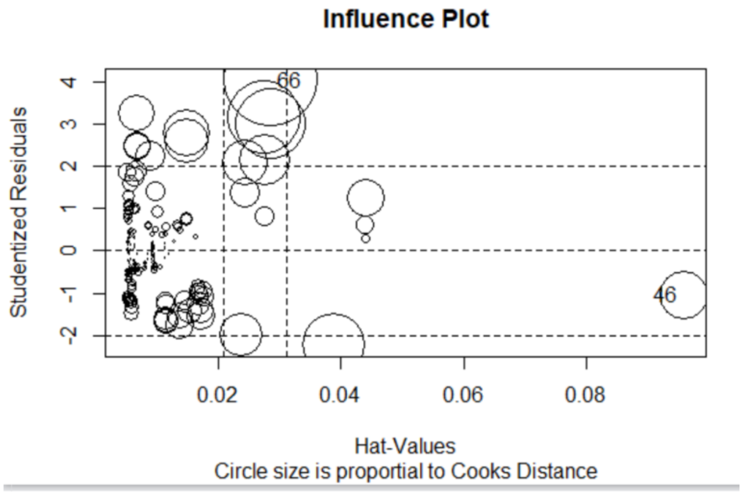
Unusual data are problematic in linear models because they can unduly influence the results of the analysis, and because their presence may be a signal that the model fails to capture important characteristics of the data.

Plotting “Cooks Distance” can be helpful in identifying influential observations - defined as the sum of all the changes in the regression model when observation is removed from it

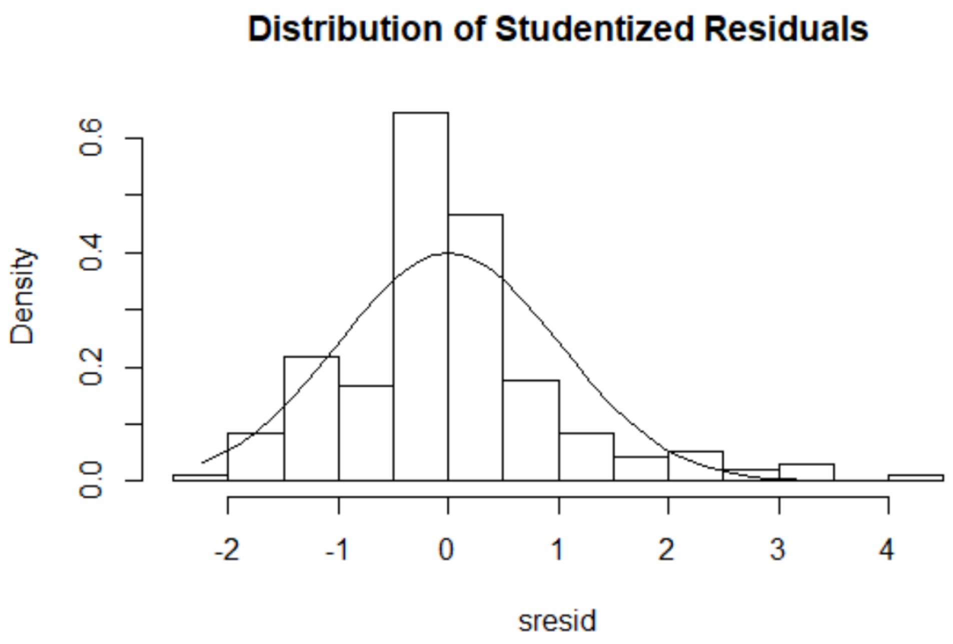


$$D_i = \frac{\sum_{j=1}^{n}\left(\hat{y}_j - \hat{y}_{j(i)}\right)^2}{ps^2}$$

```
avPlots(fit)
cutoff <- 4/((nrow(Auto) -
length((fit$coefficients)-2)))
plot(fit, which = 4, cook.levels =
cutoff)

influencePlot(fit, id.method =
'identity', main = 'Influence Plot', sub
= 'Circle size is proportial to Cooks
Distance')
```



```
# Normality of Residuals
# qq plot for studentized resid
qqPlot(fit, main="QQ Plot")
# distribution of studentized residuals
sresid <- studres(fit)
hist(sresid, freq=FALSE,
    main="Distribution of Studentized Residuals")
xfit<-seq(min(sresid),max(sresid),length=40)
yfit<-dnorm(xfit)
lines(xfit, yfit)
```



So, that’s all we’re going to cover on advanced regression diagnostics. For our purposes going forward, we’re going to stick with the basic performance metrics in AML – *unless* we run into a problem.

… understanding regression – matrix algebra

Consider an **overdetermined** system *(more equations than unknowns)*

$$\sum_{j=1}^{n} \mathrm{X}_{ij}\beta_{\mathrm{j}} = \mathrm{y}_{\mathrm{i}} \quad (i = 1,2,\dots,m)$$

Of m linear equations in n unknown coefficients β_1 , $\beta_{2,\dots}$ β_m with m > n *(note: for a linear model as above, not all of X contains information on the data points. The* ***first column is populated with ones*** $\boldsymbol{X_{i1} = 1}$ *(this is a placeholder for the intercept coefficient – the rest of the matrix values ARE coefficients, only the other columns contain actual data, and n= number of regressors + 1).* This can be written in matrix form as:

$$\mathrm{X}\beta_{\mathrm{j}} = \mathrm{y}$$

$$X = \begin{bmatrix} X_{11} & X_{12} & \cdots & X_{1n} \\ X_{21} & X_{22} & \cdots & X_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ X_{m1} & X_{m2} & \cdots & X_{mn} \end{bmatrix} \quad \beta = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_2 \end{bmatrix} \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_2 \end{bmatrix}$$

Such a system usually has no solution, so the goal is instead to fine the coefficients β *which fit the equations "best" in the sense of solving the quadratic minimization problem.*

$$\hat{\beta} = \arg_{\beta} min\, S(\beta)$$

(arg min means position where S is minimized)

… normal equations

Where the objective function S is given by:

$$S(\beta)=\sum_{j=1}^{m} |y_i - \sum_{j=1}^{n} Xij\,\beta_j|^2 = ||y - X\beta||^2$$

A justification for choosing this criterion is given in properties below. This minimization problem has a unique solution, provided that the n columns of the matrix X are ***linearly independent*** given by solving the normal equations.

$$(X^TX)\,\hat{\beta} = X^T y$$

If any column can be expressed as a linear combination of other columns, you have a dependence issue that will have to be resolved

… normal equations

The matrix X^TX is known as the Gramian matrix of X, which possesses several properties such as being a positive semi-definite matrix

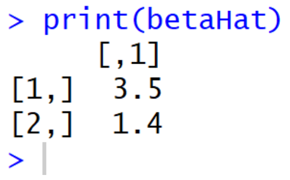
(Note: the problem is that X^TX becomes unstable when the magnitude between elements is large (often causing operations like division to interpret elements as 0 and then you get div/0 type errors – we'll see the effects of this soon)

Finally, $\hat{\beta}$ is the coefficient vector of the least-squares hyperplane, expressed as:

$$\hat{\beta} = (X^TX)^{-1} X^T y$$

Using the same matrix from the OLS example:

```
X <- cbind(1, mydata$X)
y <- mydata$Y
betaHat <- solve(t(X)%*%X) %*% t(X) %*%y
```



```
> print(betaHat)
     [,1]
[1,]  3.5
[2,]  1.4
> 
```

Note that the first column is populated with ones to hold the intercept
X <- cbind(1, mydata$X)

Computational Solutions

QR Decomposition (factorization)

Solve the linear system:

X

1	2	-1
0	1	2
0	0	1

B

a
b
c

Y

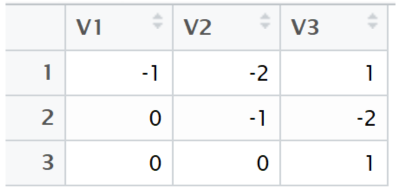
6
4
1

This is easy because we can see that c =1, which implies that b = 2, which implies that a = 3 *(backsolving) X is a triangular matrix and one of the nice things about triangular matrices is that it's easy to write a program to solve problems like this.*

We can decompose any full rank matrix X *(full rank means the columns of the matrix are independent; i.e., no column can be written as a combination of the others)* can be "factored" into "Q" matrix and an R matrix, with R being an upper triangular matrix

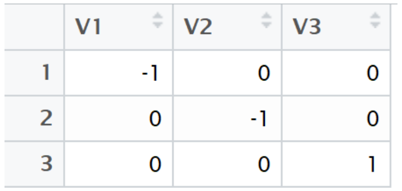
```
X <- matrix(c(1, 0, 0, 2, 1, 0, -1, 2, 1), nrow = 3)
> Y <- c(6,4,1)
> QR <- qr(X)
> Q <- qr.Q(QR)
> R <- qr.R(QR)
> betahat <- backsolve(R, crossprod(Q, Y))
➢ betahat

➢ [1,] 3
➢ [2,] 2
➢ [3,] 1
```

R



	V1	V2	V3
1	-1	-2	1
2	0	-1	-2
3	0	0	1

Q



	V1	V2	V3
1	-1	0	0
2	0	-1	0
3	0	0	1

QR Theory

Recall:	$(X^TX)\,\beta = X^T y$
Substituting the QR factors:	$(QR)^T (QR)\,\beta = (QR)^T y$
association:	$R^T (Q^TQ) R\,\beta = R^TQ^T y$
Eliminate identity *(remember, $Q^TQ = I$)*	$R^T R\,\beta = R^TQ^T y$
Multiplication Principle	$(R^T)^{-1} R^T R\,\beta = (R^T)^{-1} R^TQ^T y$
Elimination	$R\,\beta = Q^T y$

So now we're left with a much easier equation to solve

BAUER COLLEGE OF BUSINESS
UNIVERSITY of HOUSTON

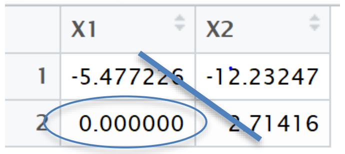
QR Example

Another example with a little more complexity:

```
setwd("C:/Users/ellen/OneDrive/Documents/Final Content/Class 1 CPM")
mydata <- read.csv(file="Ex1LS.csv", header=TRUE, sep=",")
X <- mydata[1:2]
X <- cbind(1, X) # adding the intercept placeholder
Y <- mydata[,3]
tstDF <- cbind(X, Y)
tst <- lm(Y ~ X1 + X2, mydata) # if you don't use an intercept,
# you can compare with lm by using -1 in formula to eliminate intercept
tst$coefficients
QR <- qr(X)
Q <- qr.Q(QR)
R  <- qr.R(QR)
betahat <- backsolve(R, crossprod(Q, Y))
betahat
betahat <- solve.qr(QR, Y)
betahat
```

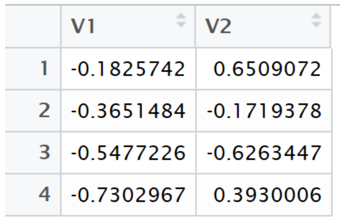
This is what the R matrix is supposed to look like

R



	X1	X2
1	-5.477226	-12.23247
2	0.000000	2.71416

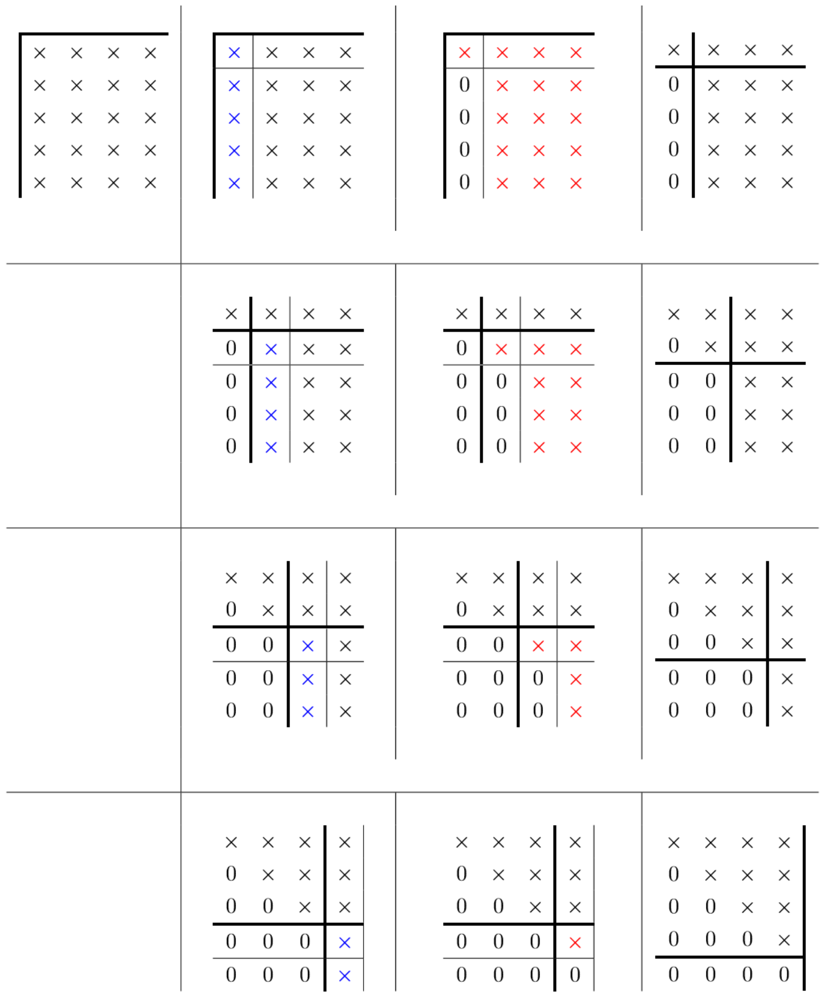
Q



	V1	V2
1	-0.1825742	0.6509072
2	-0.3651484	-0.1719378
3	-0.5477226	-0.6263447
4	-0.7302967	0.3930006

```
➢ tst$coefficients (Intercept) X1 X2 2.79850746 0.06716418 0.70149254
➢ QR <- qr(X)
➢ Q <- qr.Q(QR)
➢ R <- qr.R(QR)
➢ betahat <- backsolve(R, crossprod(Q, Y))
➢ betahat [,1] [1,] 2.79850746 [2,] 0.06716418 [3,] 0.70149254
➢ betahat <- solve.qr(QR, Y)
> betahat 1 X1 X2 2.79850746 0.06716418 0.70149254
```

QR Householder Factorization



QR Householder factors a matrix into 2 matrices, Q and R.

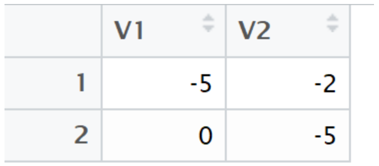
QRH decomposition is often used to solve the linear least squares problem, and is the basis for a particular eigenvalue algorithm, the QR algorithm

The QR process iteratively eliminates all the elements in a column in a triangular pattern – with each column in the R matrix having ncol-1 non-zero elements left.

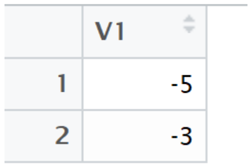
1. Robert A. van de Geijn University of Texas at Austin

QRH Factorization

After running the data from the handout through, the algorithm produces the following R and Y matrix



	V1	V2
1	-5	-2
2	0	-5



	V1
1	-5
2	-3

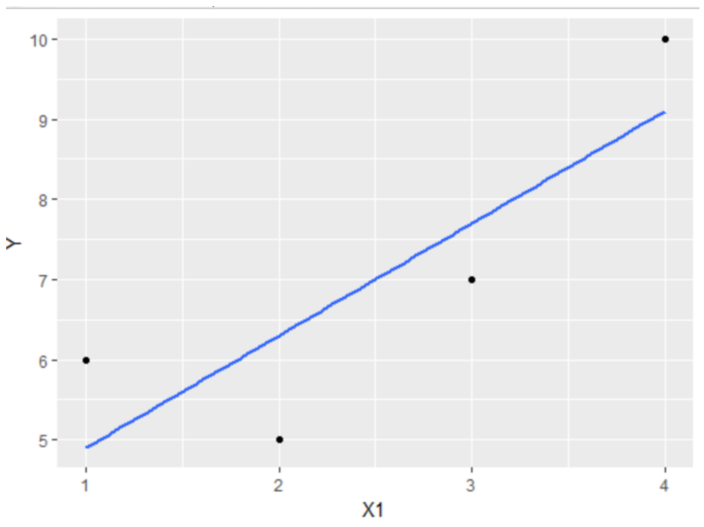
Which now should be easy using backsolve

-3 / -5 = 0.6
(-5-(-2*0.6))/-5 = 0.76

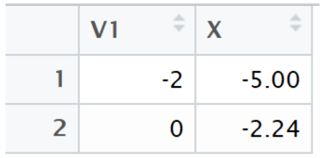
QRH Regression

```
setwd("C:/Users/ellen/OneDrive/Documents/Final Content/Class 1
CPM")
mydata <- read.csv(file="Ex1LS.csv", header=TRUE, sep=",")
model <- lm( Y ~ X1 ,mydata)
model$coefficients

p <- ggplot(data = mydata, aes(x= X1, y= Y)) + geom_point() +
geom_smooth(method = 'lm', se = FALSE)
p

Y <- mydata$Y
X <- mydata$X1
X <- as.matrix(cbind(1, X))

res <- QR.regression(Y, X)

res$beta
```

```
➢ model$coefficients
➢ (Intercept) X1
➢ 3.5 1.4
➢ Y <- mydata$Y
➢ X <- mydata$X1
➢ X <- as.matrix(cbind(1, X))
➢ res <- QR.regression(Y, X)
➢ > res$beta
➢ [1] 3.5 1.4
```



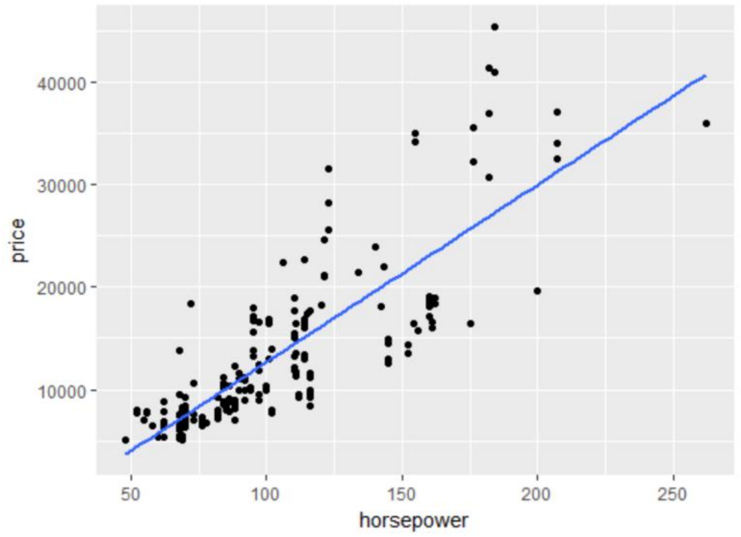
R



	V1	X
1	-2	-5.00
2	0	-2.24

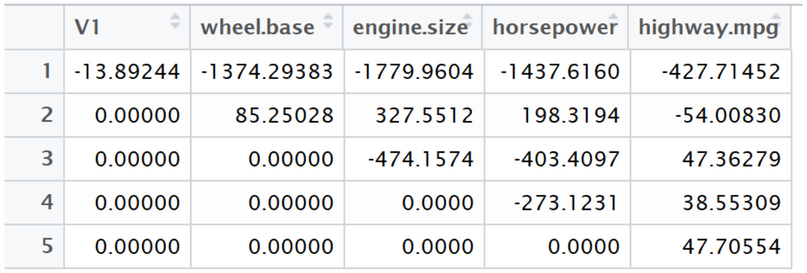
QRH Regression

```
setwd("C:/Users/ellen/OneDrive/Documents/Final Content/Class 1
CPM")
mydata <- read.csv(file="Ex1LS.csv", header=TRUE, sep=",")
X <- mydata[1:2]
X <- cbind(1, X) # adding the intercept placeholder
Y <- mydata[,3]
tstDF <- cbind(X, Y)
tst <- lm(Y ~ X1 + X2, mydata) # if you don't use an intercept,
# you can compare with lm by using -1 in formula to eliminate intercept
tst$coefficients

p <- ggplot(data = mydata, aes(x= X1, y= Y)) + geom_point() +
geom_smooth(method = 'lm', se = FALSE)
p

X <- as.matrix(X)
```



```
➤ model$coefficients (Intercept) wheel.base engine.size
  horsepower highway.mpg
➤ -32094.29199 230.89312 109.39590 65.81011 55.60859
> Y <- Autos$price
➤ X <- Autos[1:4]
➤ X <- as.matrix(X)
➤ X <- as.matrix(cbind(1, X))
➤ res <- QR.regression(Y, X)
➤ res$beta [1]
➤ -32094.29199 230.89312 109.39590 65.81011 55.60859
```

R



	V1	wheel.base	engine.size	horsepower	highway.mpg
1	-13.89244	-1374.29383	-1779.9604	-1437.6160	-427.71452
2	0.00000	85.25028	327.5512	198.3194	-54.00830
3	0.00000	0.00000	-474.1574	-403.4097	47.36279
4	0.00000	0.00000	0.0000	-273.1231	38.55309
5	0.00000	0.00000	0.0000	0.0000	47.70554

QRH Regression *(handout on blackboard)*

```
X <- matrix(c(3, 0, 4, -2, 3, 4), nrow = 3, ncol = 2)
y <- matrix(c(3, 5, 4), nrow = 3, ncol = 1)

# Householder Function

 nr <- length(y)
 nc <- NCOL(X)

 for (j in seq_len(nc))
{
   id <- seq.int(j, nr)
   sigma <- sum(X[id,j]^2)
   s <- sqrt(sigma)
   diag_ej <- X[j,j]
   gamma <- 1.0 / (sigma + abs(s * diag_ej))
   kappa <- if (diag_ej < 0) s else -s
   X[j,j] <- X[j,j] - kappa
   if (j < nc)
     for (k in seq.int(j+1, nc))
     {
       yPrime <- sum(X[id,j] * X[id,k]) * gamma
       X[id,k] <- X[id,k] - X[id,j] * yPrime
     }
   yPrime <- sum(X[id,j] * y[id]) * gamma
   y[id] <- y[id] - X[id,j] * yPrime
   X[j,j] <- kappa
 }
RX <- as.matrix(X[1:ncol(X), ])
Ry <- as.matrix(y[1:ncol(X),])
```

(1) let z = the first column of the submatrix B, where B $= \widehat{A}_{k:m,k:n+1}$

(2) Construct a Householder transformation that zeros out z below the first entry in z:

  (a) $v = \text{sign}(z_1)||z||_2\, e + z$ %vector normal to a Householder "mirror"

  (b) $v = v/||v||_2$ % unit vector normal to a Householder "mirror"

$$(1)\ B = \widehat{A} = \begin{pmatrix} 3 & -2 & 3 \\ 0 & 3 & 5 \\ 4 & 4 & 4 \end{pmatrix},\ z = \begin{pmatrix} 3 \\ 0 \\ 4 \end{pmatrix}.$$

$$(2)\ (a)\ v = \text{sign}(z_1)||z||_2\, e + z = \text{sign}(3)\,5 \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 3 \\ 0 \\ 4 \end{pmatrix} = (+1)\,5 \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 3 \\ 0 \\ 4 \end{pmatrix} = \begin{pmatrix} 8 \\ 0 \\ 4 \end{pmatrix}.$$

norm

z_1

Vector e is combined in this operation below

For z in step 1), let sign(z_1) be +1 if $z_1 > 0$ and let sign(z_1) be -1 if $z_1 < 0$. z_1 is the first component of z. Also let e be a vector of the same dimension as z that is all zero except the first element is one. Here are details for the above algorithm:

Note: you will not be responsible for the details of this algorithm – just understand how it works (and for those who are curious and want to know the details)

QRH wrap-up

- Again, you won’t be tested on QRh code, but you will be asked about the theory.
- QR is a very stable approach to solving a wide range of linear regression problems: including non-parametric, non-linear and multivariate models. An alternative is Gradient Descent, which has more flexibility and is computationally faster - the next section.

Gradient Descent

We're going to spend some time on gd - it is common throughout machine learning as an optimization component – from linear regression to neural networks.

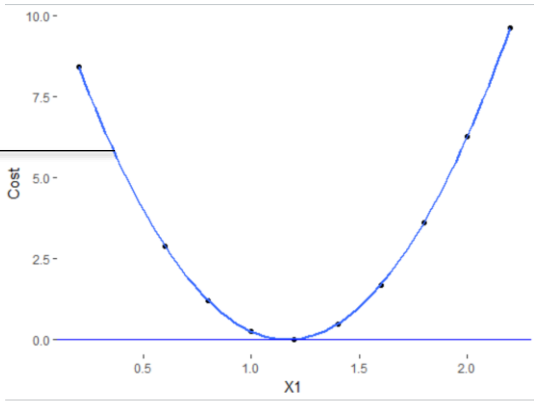
Recall:

Analytic Solutions:

Cost Function (assumed to be convex)

$$\frac{\partial s}{\partial \beta_2} = 0 = 20\beta_1 + 60\beta_2 \; - 154 \; = 1.4$$

Holding β_1 constant



OLS becomes intractable in the real world

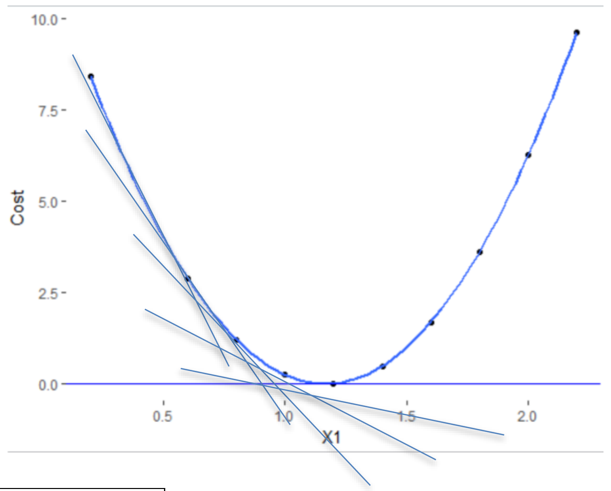
Cost function canonical expression $$\frac{1}{2m}\sum_{i=1}^{m}(f_\theta(x_i) - y_i)^2$$

θ theta is a vector $[\theta_1, \theta_2, \ldots. \theta_n]$ that holds the parameters *(e.g., intercept$_0$+ slope$_1$ + slope$_2$…)*

note: $\frac{1}{2m}$ *is not necessary to minimize, but is used to average the error so that models are more comparable (and to make derivative computation easier)*

$$J(\theta_1, \theta_2) = \frac{1}{2}(f_\theta(xi) - yi)2 = \frac{1}{2}\,(\theta_1 + \theta_2 x) - y)2$$

$$\frac{\partial J}{\partial \theta_2}\ \frac{1}{2}\,(\theta_1 + \theta_2 x) - y)2 = x\,((\theta_1 + \theta_2 x) - y) = x\,(\hat{y} - y)$$



slope of our OLS solutions - tangents to cost function with intercept held constant (3.5)

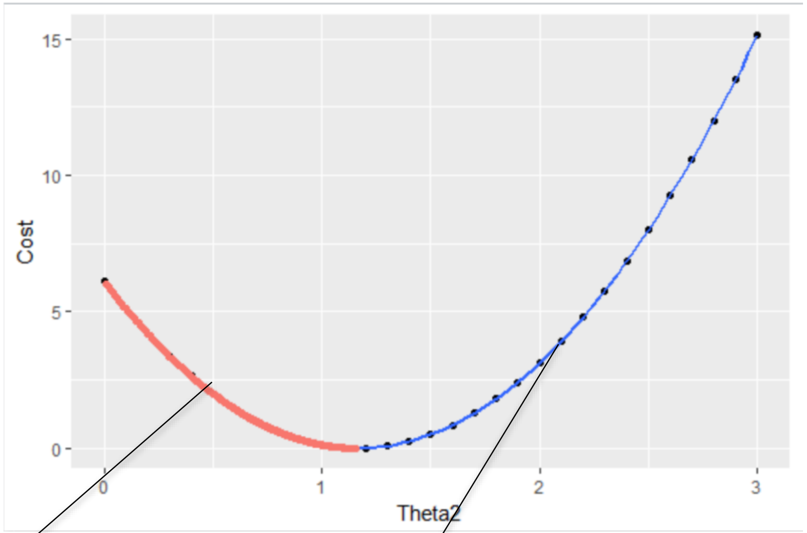
X	Y	theta1	theta2	Yhat = theta1 + (theta2*X)	SSE = 1/2(Y-Yhat)^2	*d* SSE/*d* theta2 = -(Y-Yhat)X
3	7	3.5	0.0	3.5	6.125	-10.5
3	7	3.5	0.2	4.1	4.205	-8.7
3	7	3.5	0.4	4.7	2.645	-6.9
3	7	3.5	0.6	5.3	1.445	-5.1
3	7	3.5	0.8	5.9	0.605	-3.3
3	7	3.5	1.0	6.5	0.125	-1.5
3	7	3.5	1.2	7.1	0.005	0.3
3	7	3.5	1.4	7.7	0.245	2.1
3	7	3.5	1.6	8.3	0.845	3.9
3	7	3.5	1.8	8.9	1.805	5.7
3	7	3.5	2.0	9.5	3.125	7.5
3	7	3.5	2.2	10.1	4.805	9.3

*Note that slope turns positive ~ 1.4 (imprecision due to really large **alpha** and rounding)*

So how do we figure out where the slope turns positive (and stop - that's our objective)?

Gradient descent attempts to minimize the error by solving a sequence of smaller minimization problems

$$\nabla J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(f_\theta(x_i) - y_i\right)^2$$



```
cost2 <- function(X, y, theta) {
  sum ((((X %*% theta)+Intercept) - y)^2 ) / (2*length(y))
}
```

1. Map out the cost function which measures MSE

2. Then, walk along the gradient until it turns positive.

BAUER
COLLEGE OF BUSINESS
UNIVERSITY of HOUSTON

Simple GD Function (not production ready – for teaching purposes)

```
alpha <- 0.001
num_iters <- 500
theta <- 0
chartPts <- NULL
epsilon <- .12

tracerMat <- matrix(numeric(0), nrow = num_iters, ncol = 6)

for (i in 1:num_iters) {
  error <- ((X %*% theta)+Intercept - y) # add fixed intercept
  delta <- t(X) %*% error / length(y)
  theta <- theta - (alpha * delta)
  cost <- cost2(X, y, theta)
  chartPts <- rbind(chartPts, c(theta, cost))
  if ((sqrt(sum(delta^2)))<=epsilon) {break}
  tracerMat[i,] <- c(i, error, delta, theta, cost, sqrt(sum(delta^2)))
}
```

a - alpha sets the increments or learning rate (how far to step along the gradient)

Iterations is to ensure you don't get into a infinite loop in case there's no convergence

e - epsilon is often called the threshold for the optimizer convergence – it sets the sensitivity, or range of error you're willing to accept

This is the derivative… the gradient $\frac{1}{2m}\sum_{i=1}^{m}(f_\theta(xi) - yi)^2$

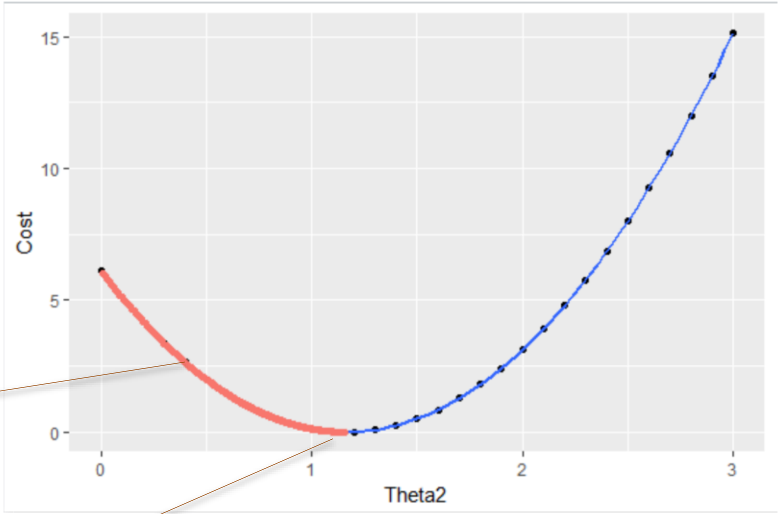
This determines the next step based on the current gradient, the alpha (note that it subtracts – so a negative slope will become positive (and vice-versa) which assures that the algorithm will constantly seek the minimum

This tests to see if the gradient step is less than the threshold – if so, then it breaks out of the loop

The other lines just record data for graphing and learning

You will be asked to apply this function to a problem on the exam

```
for (i in 1:num_iters) {
  error <- ((X %*% theta)+Intercept - y) # add fixed intercept
  delta <- t(X) %*% error / length(y)
  theta <- theta - (alpha * delta)
  cost <- cost2(X, y, theta)
  chartPts <- rbind(chartPts, c(theta, cost))
  if ((sqrt(sum(delta^2)))<=epsilon) {break}
  tracerMat[i,] <- c(i, error, delta, theta, cost, sqrt(sum(delta^2)))
}
```



It just *slides* down the function until it hits bottom (within a tolerance setting) or the number of iterations runs out

convergence

i	error	delta	theta	cost	sqrt(delta)^2
1	-3.5000	-10.5000	0.0105	6.0152	10.5000
2	-3.4685	-10.4055	0.0209	5.9075	10.4055
3	-3.4373	-10.3119	0.0312	5.8016	10.3119
4	-3.4063	-10.2190	0.0414	5.6976	10.2190
5	-3.3757	-10.1271	0.0516	5.5955	10.1271
6	-3.3453	-10.0359	0.0616	5.4953	10.0359
7	-3.3152	-9.9456	0.0715	5.3968	9.9456
8	-3.2854	-9.8561	0.0814	5.3001	9.8561
9	-3.2558	-9.7674	0.0912	5.2051	9.7674
494	-0.0406	-0.1218	1.1533	0.0008	0.1218
495	-0.0402	-0.1207	1.1534	0.0008	0.1207
NA	NA	NA	NA	NA	NA

Notice that, as the algorithm runs, error, and delta decrease (theta increases). The delta determines the direction (if delta is negative the GD adds the delta – if delta is positive, GD subtracts the delta and the direction reverses (that didn't happen in this example because we set epsilon to break before that happened.

The previous example used just one theta value to illustrate. We're now moving theta vectors.

```
library(plotly)

setwd("C:/Users/ellen/OneDrive/DA2/Regression/Data")
mydata <- read.csv(file="Ex1LS-2-BU.csv", header=TRUE, sep=",")

x <- as.matrix(mydata[,1])
y <- mydata[,3]

lmMod <- lm(Y ~ X1, mydata)
lmMod$coefficients

# squared error cost function
cost <- function(X, y, theta) {
  sum( (X %*% theta - y)^2 ) / (2*length(y))
}

# learning rate and iteration limit

alpha <- 0.1
num_iters <- 1000
epsilon <- .001

# keep history
cost_history <- double(num_iters)
theta_history <- list(num_iters)

# initialize coefficients
#theta <- matrix(c(0,0,0), nrow=3)
theta <- rep(0, ncol(x)+1)

# add a column of 1's for the intercept coefficient
X <- cbind(1, x)
```

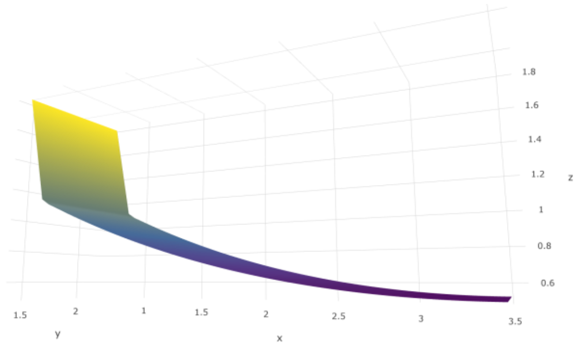
You have to add a column of 1's as a placeholder for the intercept

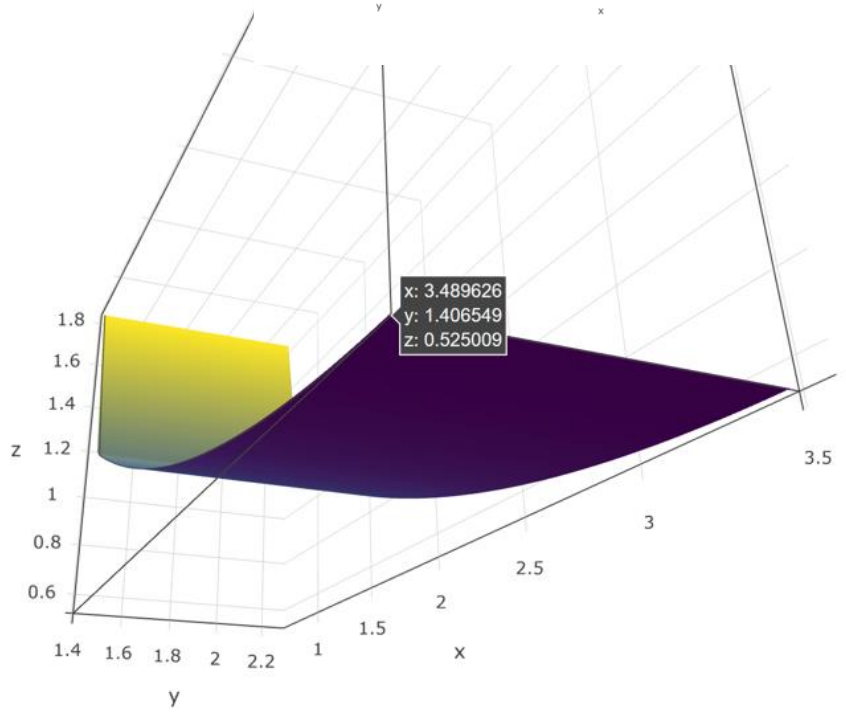
```
# gradient descent
for (i in 1:num_iters) {
  error <- (X %*% theta - y)
  delta <- t(X) %*% error / length(y)
  theta <- theta - alpha * delta
  cost_history[i] <- cost(X, y, theta)
  theta_history[[i]] <- theta
  if ((sqrt(sum(delta^2)))<=epsilon) {break}
}

tst <- data.frame(cost_history)

round(theta, 1)

tM <- matrix(unlist(theta_history),ncol=2,byrow=TRUE)
Intercept <- tM[,1]
Slope <- tM[,2]
zM1 <- as.matrix(cost_history)


# ----------------------- graphing 3 variables -----------#


Cost <- matrix(0,length(Intercept), length(Slope));
for(i in 1:length(Intercept)){
  for(j in 1:length(Slope)){
    t1 = c(Intercept[i],Slope[j])
    Cost[i,j]= zM1[j]
  }
}

dfSurface1 <- data.frame(Cost)
zM1 <- as.matrix(dfSurface1)

p <- plot_ly (x =  Intercept, y = Slope, z = zM1) %>% add_surface()
p
```

epsilon controls "tolerance" – too large and it won't optimize, too small and it won't converge

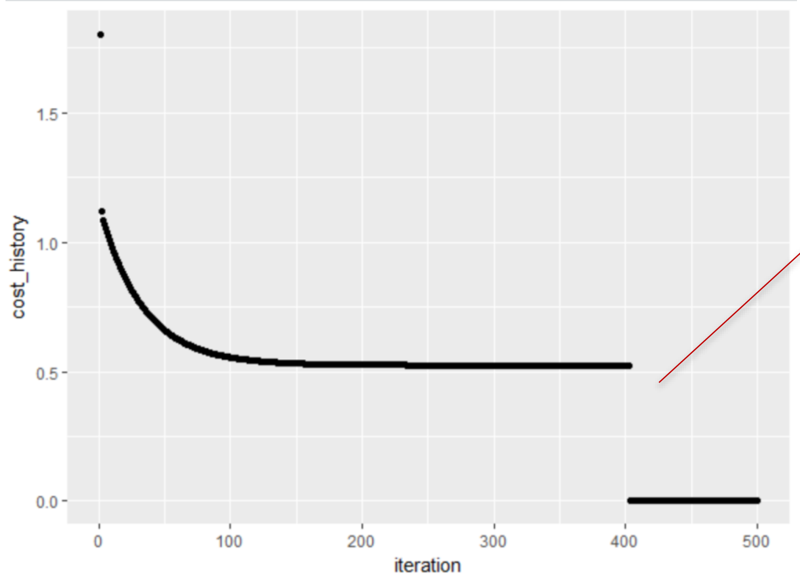
```
➢ lmMod$coefficients
➢ (Intercept) X1 3.5 1.4
```

```
round(theta, 1)
[1,] 3.5
[2,] 1.4
```





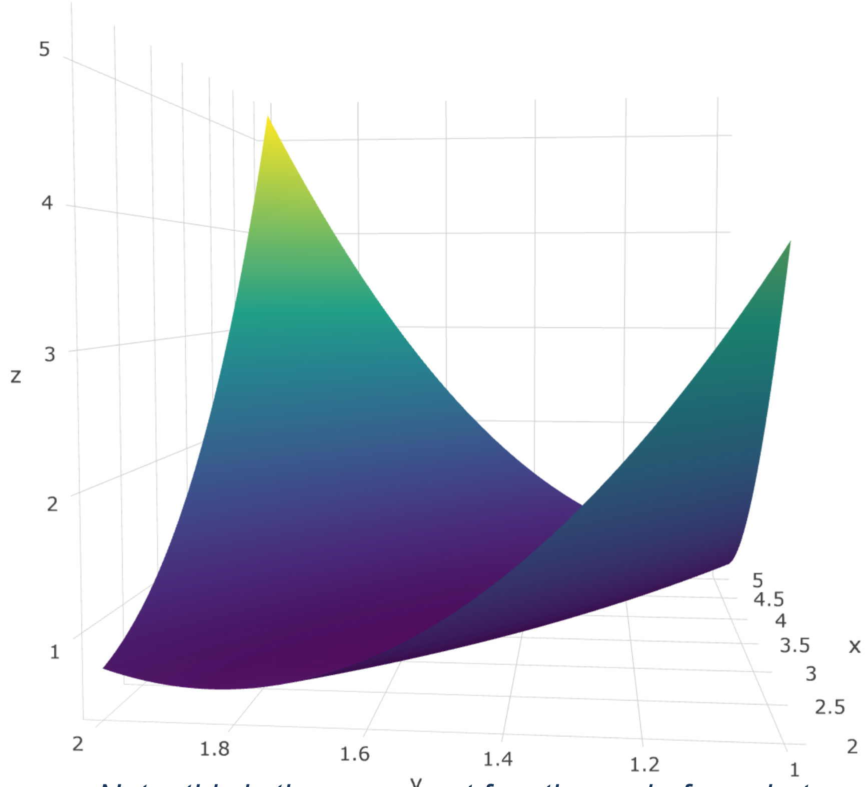
Convergence plot



Delta hit epsilon

To an earlier point, tolerance can be adjusted to a minimum or eliminated altogether, however the optimization must be able to reach convergence or the algorithm will run endlessly

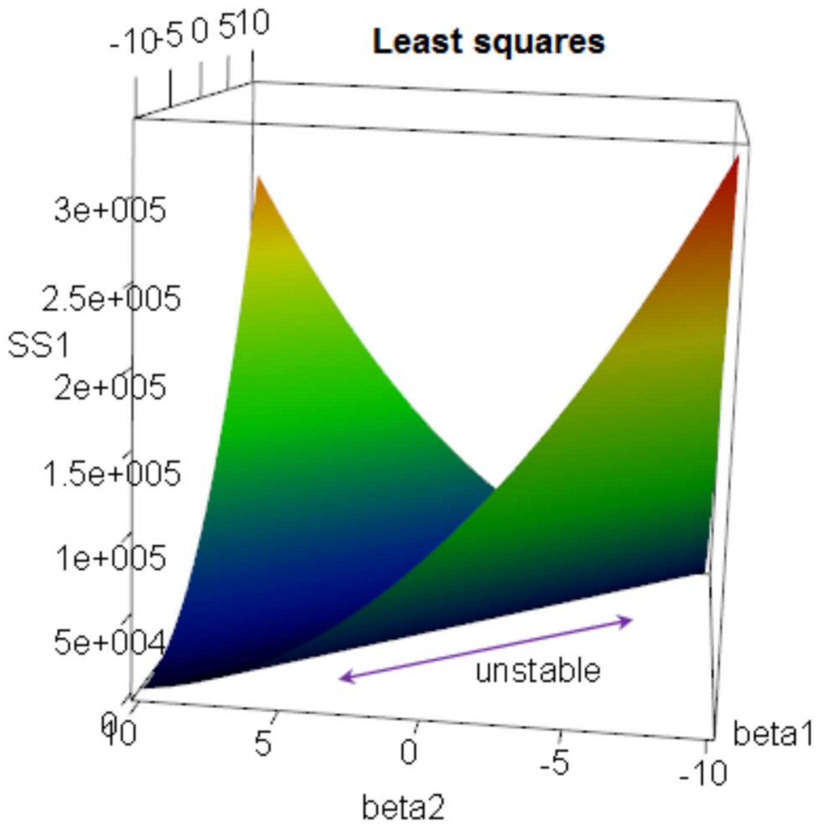
```
Intercept1 <- seq(from=(round((theta[1])-(theta[1]*.5)))
                  ,to=(round((theta[1])+(theta[1]*.5)))
                  ,length=num_iters)

Slope1 <- seq(from=(round((theta[2])-(theta[2]*.5)))
              ,to=(round((theta[2])+(theta[2]*.5)))
              ,length=num_iters)


Cost1 <- matrix(0,length(Intercept1), length(Slope1));
for(i in 1:length(Intercept1)){
  for(j in 1:length(Slope1)){
    t1 = c(Intercept1[i],Slope1[j])
    Cost1[i,j]= cost(X, y, t1)
  }
}

dfSurface1 <- data.frame(Cost1)
zM1 <- as.matrix(dfSurface1)

p <- plot_ly (x =  Intercept1, y = Slope1, z = zM1) %>% add_surface()
p
```

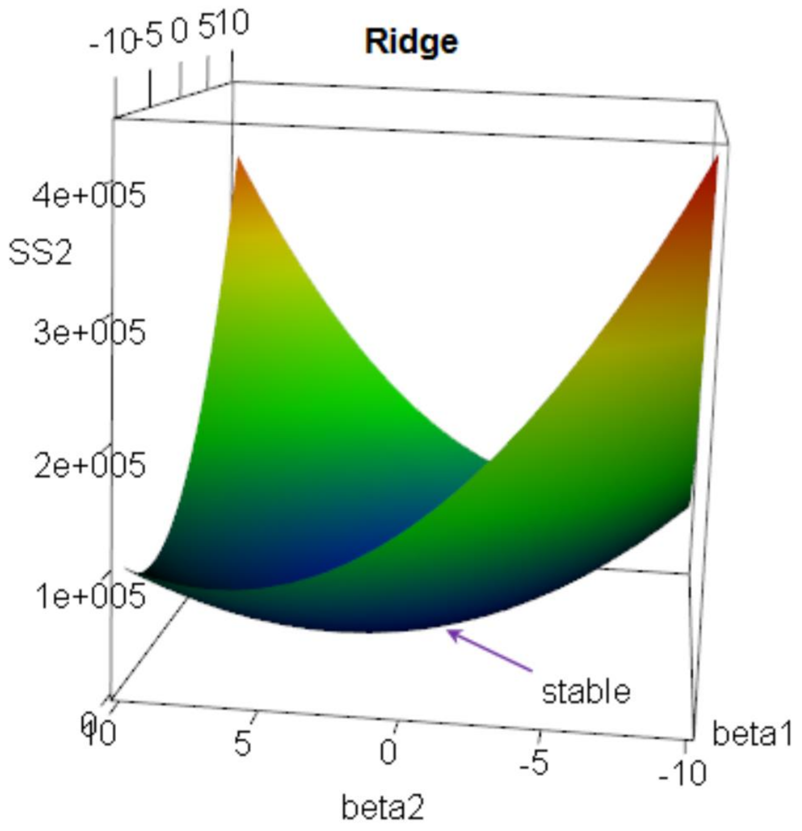


Note: this is the same cost function as before - but we plotted the complete shape using a wider range of thetas (GD picks one point and slides from there – this is the cost curve without alpha and epsilon driving)

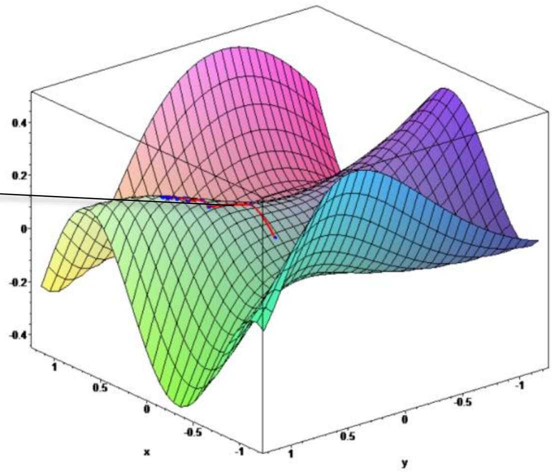
The takeaway here is understanding how gd guides the journey of finding the minimum cost ***using a least squares cost function***

Later, we'll add a ***regularization term*** to stabilize the minimum, which also generalizes the model *(to decrease variance and avoid overfitting)* so that it will adapt to new data





Cost functions can be complex *(really complex – remember, we're often dealing in high dimensional space with kernel values),* with many challenges *(e.g., saddle points)*



These are the challenges of optimization and there are many approaches to deal with it: derivative free, particle swarm algorithms…. But there's always that trade-offs:

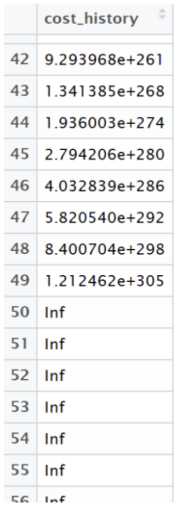
- Interoperability vs. Flexibility
- Bias vs. Variance

```
➢ lmMod$coefficients (Intercept) Education Seniority
➢ -50.0856387 5.8955560 0.1728555 >
```

```
# ---------------- More Complex Dataset --------------#

setwd("C:/Users/ellen/OneDrive/DA2/Regression/Data")
incomeData <- read.csv(file="Income2.csv", header=TRUE, sep=",")

x <- as.matrix(cbind(incomeData[,2:3]))
y <- incomeData$Income

lmMod <- lm(Income ~ Education + Seniority, incomeData)
lmMod$coefficients
```

(this is the same)

```
# gradient descent
for (i in 1:num_iters) {
  error <- (X %*% theta - y)
  delta <- t(X) %*% error / length(y)
  theta <- theta - alpha * delta
  cost_history[i] <- cost(X, y, theta)
  theta_history[[i]] <- theta
  if ((sqrt(sum(delta^2)))<=epsilon) {break}
}

tst <- data.frame(cost_history)
```



	cost_history
42	9.293968e+261
43	1.341385e+268
44	1.936003e+274
45	2.794206e+280
46	4.032839e+286
47	5.820540e+292
48	8.400704e+298
49	1.212462e+305
50	Inf
51	Inf
52	Inf
53	Inf
54	Inf
55	Inf
56	Inf

```
Error in if ((sqrt(sum(delta^2))) <=
epsilon) { : missing value where
TRUE/FALSE needed
```

This *tells us a value has bombed out*

Taking a look at cost history – it was increasing into infinity.

A production GD algorithm is designed to scale the data before processing.

Let’s look at how that works:

BAUER COLLEGE OF BUSINESS UNIVERSITY of HOUSTON

… gd and scaling

```
sX <- as.matrix(cbind(1, scale(X[,2:3])))
sy <- scale(y)


alpha <- 0.1
num_iters <- 1000
epsilon <- .001

# keep history
cost_history <- double(num_iters)
theta_history <- list(num_iters)

# initialize coefficients
#theta <- matrix(c(0,0,0), nrow=3)
theta <- rep(0, ncol(x)+1)

# add a column of 1's for the intercept coefficient
X <- sX
y <- sy
# gradient descent
for (i in 1:num_iters) {
  error <- (X %*% theta - y)
  delta <- t(X) %*% error / length(y)
  theta <- theta - alpha * delta
  cost_history[i] <- cost(X, y, theta)
  theta_history[[i]] <- theta
  #  if ((sqrt(sum(delta^2)))<=epsilon) {break}
}

tst <- data.frame(cost_history)

theta
```

Scale the data and run it again

```
theta
[,1] 3.285335e-16
Education 8.316551e-01
Seniority 3.565190e-01
```

So now it works, but what can we do with those teeny scaled numbers?

… gd backscaling

```
sMod <- lm(y ~ X[,2] + X[,3])
sMod$coefficients

# of course, you don't want scaled predictions.
# a number of ways we can deal with that
# you can run a model and backscale the results

# scale the original data again, without the intercept placeholer

sdata <- scale(incomeData[,2:4])
# Save scaled attibutes
scaleList <- list(scale = attr(sdata, "scaled:scale"),
              center = attr(sdata, "scaled:center"))


# convert the scaled data to a dataframe and run it through lm
sdata <- as.data.frame(sdata)
smod <- lm(Income ~ Education + Seniority, data = sdata)

# create scaled predcitions for Income
sp <- predict(smod, sdata)

# Fit the same model to the original cars data:
omod <- lm(Income ~ Education + Seniority, data = incomeData)
op <- predict(omod, incomeData)

# Convert scaled prediction to original data scale:
usp <- sp * scaleList$scale["Income"] + scaleList$center["Income"]

# Compare predictions:
all.equal(op, usp)
# note that this all works because the method (QR) is the same
```

```
# so, now we've backscaled the predictions and that works
# let's try a different approach and backscale the coefficients

X <- incomeData[,2:3]
Y <- incomeData[,4]

library(R1magic)
betas.scaled <- matrix(smod$coefficients)
betas <- scaleBack.lm(X, Y, betas.scaled)
betas


# compare to original model
omod$coefficients


tM <- matrix(unlist(theta_history),ncol=2,byrow=TRUE)
Intercept <- tM[,1]
Slope <- tM[,2]
zM1 <- as.matrix(cost_history)
```

```
betas <- scaleBack.lm(X, Y, betas.scaled) > betas [,1] -
50.0856387 [2,] 5.8955560 [3,] 0.1728555 > > > # compare
to original model > omod$coefficients (Intercept)
Education Seniority
-50.0856387 5.8955560 0.1728555 >
```

Scaling, Standardizing, Normalizing

Scaling (rescaling) a vector is adding or subtracting a constant and then multiplying or dividing by a constant to change the units of data (e.g., Convert from Celsius to Fahrenheit).

Normalizing a vector is usually dividing by a norm of the vector (e.g., making the Euclidian length equal to one). Often means rescaling by the minimum and range of the vector, to make all the elements between 0 ⇔ 1.

Standardizing a vector is usually subtracting a measure of location and dividing by a measure of scale (e.g., if the vector has a Gaussian distribution, then subtract the mean and divide by the standard deviation), again resulting in all the elements between 0 ⇔ 1.

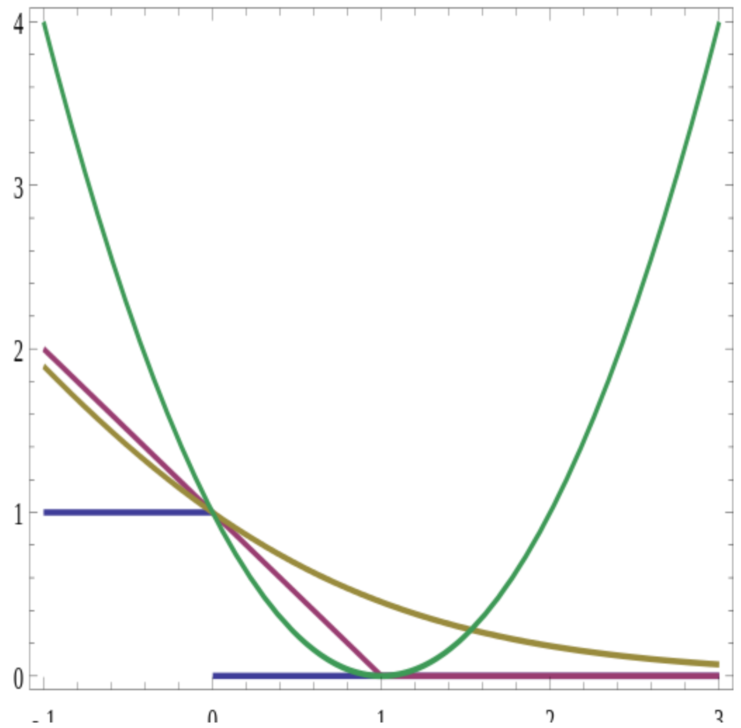
… gd

Closing thoughts

Cost functions and regularization terms are central to machine learning. Gradient descent is present in regression, neural networks and others.

Algorithms use different cost functions and new ones(or variations) are created often.

We will get into hinge loss in Classification.



- 0–1 indicator function
- Square loss function
- Hinge loss function.
- Logistic loss function.